SAND REPORT

SAND2004-6541
Unlimited Release
Printed January 2005

Supersedes SAND2002-2775
dated October 2002

ALEGRA: Version 4.6

S. K. Carroll, R. R. Drake, D. M. Hensinger,
C. B. Luchini, S. V. Petney, J. Robbins,
A. C. Robinson, R. M. Summers, T. E. Voth, M. K. Wong,
T. A. Brunner, C. J. Garasi, T. A. Haill and T. A. Mehlhorn

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2004-6541
Unlimited Release
Printed January 2005

Supersedes SAND2002-2775
dated October 2002

ALEGRA: Version 4.6

S. K. Carroll, R. R. Drake, D. M. Hensinger,
C. B. Luchini, S. V. Petney, J. Robbins,
A. C. Robinson, R. M. Summers, T. E. Voth, and M. K. Wong
Computational Physics Research & Development

T. A. Brunner, C. J. Garasi, T. A. Haill and T. A. Mehlhorn
HEDP Theory and ICF Target Design

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0378

Abstract

ALEGRA is an arbitrary Lagrangian-Eulerian multi-material finite element code used for modeling solid dynamics problems involving large distortion and shock propagation. This document describes the basic user input language and instructions for using the software.

Acknowledgment

Many individuals have contributed to ALEGRA development over many years. George Allshouse (deceased),who made important contributions to the early planning of the ALEGRA project and facilitated its beginning. Mike McGlaun launched the code development effort and Keith Matzen and Tim Trucano also contributed to early planning activities for the project.

Below we give major contributors to the ALEGRA code and attempt to list some specific areas of the current active code base to which they have contributed. Many of these contributers will have made significant contributions to the text of this manual. The authors of this document represent current developers actively backing this release.

Ray Bell	`SMYRA` Material Interface Tracker
Ed Boucheron	Mesh adaptivity, Desktop tools
Rebecca Brannon	Material models
Kevin Brown	Contact algorithms (`ACME`), Architecture, Solid Mechanics
Tom Brunner	Advanced Physics
Kent Budge	Architecture, Lagrangian physics, Advanced physics, Penetration
Shawn Burns	Verification, Applications
Dan Carroll	ALE algorithms, Applications
Sue Carroll	Configuration management, Regression testing, Support tools
Mary Chen	Verification & Validation
Mark Christon	Structured Mesh Development
Kyle Cochrane	Advanced Physics
Rich Drake	NEVADA tools & code infrastructure
Archie Farnsworth	Electromechanics modeling, Applications
Grant Farnsworth	Solid kinematics
Chris Garasi	Verification & Validation, Advanced Physics Applications
David Hensinger	Structured Mesh, I/O
Thomas Haill	Advanced physics
David Ketcheson	Quaternion Solid kinematics
Will McLendon	Advanced physics
James Peery	Architecture, ALE algorithms, Parallelization

Sharon Petney	Mesh material insertions, Material models, Infrastructure
Josh Robbins	Electromechanical models, Applications
Allen Robinson	Resistive MHD, Electromechanics, Parallelization
Randy Summers	Contact algorithms, Platform Adaptation, Advanced Physics
Tim Trucano	Verification & Validation, Applications
Thomas Voth	Penetration mechanics, `ACME` Support
Randy Weatherby	Mesh adaptivity, Desktop tools, Visualization
Mike Wong	Mesh adaptivity, Material models

In addition to the above list, other colleagues at Sandia and elsewhere have provided guidance and assistance with respect to ALEGRA development, including Steve Attaway, Pang Chen, Ben Cole, Dave Crawford, Doug Drumheller, Jonathan Hu, Frank Mello, Steve Montgomery, Judy Sturtevant, Bryan Oliver and Ray Tuminaro. Others have utilized portions of ALEGRA as a framework, now currently know as NEVADA, for developing special purpose capabilities, including Mike Glass, Steve Kempka, Josh Robbins, Rob Schmitt, Dave Turner, Mike Pasik, Dave Siedel, Bill Bohnhoff, Jennifer Powell. A number of users of the code have provided very valuable feedback to the development team that helped improve the code enormously, including John Aidun, Mark Boslough, Lalit Chhabildas, Paul Demmie, Jeff Lawrence, Rich Jensen, Steve Schraml, Dan Mosher, Dan Kletzli, Scott Wunsch, Ray Lemke, Bob Campbell. Many students have helped with various aspects of the code and with the development of test suites including Mary Chen, Rachel Bixler, Elena Agustin.

This page intentionally left blank.

Contents

List of Figures

List of Tables

Summary

In 1990 an effort was launched at Sandia National Laboratories to develop a state-of-the-art code that combined the modeling features of modern Eulerian shock codes, such as CTH, with the improved numerical accuracy of modern Lagrangian finite element codes. The resulting code, called ALEGRA, uses an arbitrary Lagrangian-Eulerian (ALE) formulation on an unstructured finite element mesh. This formulation allows the user to designate whether material should flow through a stationary mesh (pure Eulerian), whether the mesh should move with the material (pure Lagrangian), or whether the mesh should move independently from the material motion (arbitrary). The latter capability permits a calculation to proceed in Lagrangian fashion until the mesh becomes too distorted. At that time, mesh points in the most deformed portion of the mesh are moved to reduce the distortion to acceptable levels. The advantage is that numerical dissipation is avoided until large deformations occur and then is limited to only those regions where there are severe mesh distortions and the mesh must be moved.

ALEGRA is written predominantly in the C++ programming language Object-oriented programming techniques can be used to manage the inherent complexity of the physics being modeled. However, various Fortran-based models and libraries are also incorporated if they are sufficiently mature and robust. In many cases there is little advantage to rewriting such software.

ALEGRA has been designed to run on distributed-memory parallel computers. This was done because the enormous memories and processor speed of massively parallel processor (MPP) computers are needed to analyze large, three-dimensional problems. The memory requirements for ALEGRA scales inversely with the cube of the cell size. For example, if one halves the cell size in each direction, then the memory requirement increases by a factor of eight. ALEGRA uses explicit time integration schemes so the time step scales proportionally to the cell size. For example, if one halves the cell size, the code cuts the time step in half. Therefore, the Floating Point OPerations (FLOPs) scale inversely as the fourth power of the cell size.

The database of a large, three-dimensional problem is too large to fit on any single compute node. Therefore, ALEGRA was designed with the Single Program Multiple Data (SPMD) paradigm, in which the mesh is decomposed into sub-meshes so that each processor gets a single sub-mesh with approximately the same number of elements. Good mesh decomposition is important

to minimize the data passed between compute nodes. Whereas rectangular meshes are relatively easy to decompose, subdividing the arbitrary connectivity meshes used by ALEGRA is much more difficult. Specialized software such as the CHACO package developed at Sandia is used to decompose these meshes.

ALEGRA uses one layer of ghost elements around the sub-mesh perimeter for sub-domain boundary conditions. The database for the ghost elements must be updated during the computational cycle by interprocessor communication. These additional ghost elements represent a parallel processing cost that can be quite large for compute nodes with a small number of elements. For example, a cube meshed in a 10 by 10 by 10 regular pattern, approximately half the elements are boundary elements. For large, roughly cubic meshes, the fraction of boundary elements goes as approximately $6/N^{1/3}$, where N is the number of elements, so a million-element mesh will include only about 6% boundary elements. The trade-offs between computation, communication and memory must be actively managed by the user.

ALEGRA also provides a mesh refinement capability that in principle can provide high precision simulations without the computational cost of using a highly resolved mesh everywhere. This capability may be important for simulations over large spatial domains that require high precision in the presence of certain localized features. This capability, however, comes at a high cost in terms of memory layout and a corresponding impact on efficiency and this cost also impacts non-adaptive calculations. Adaptive capability should not be considered under active development with a high priority support status during the release period which this manual represents.

The multi-block curvilinear structured (MBS) mesh option in ALEGRA combines the speed and footprint benefits available to a structured code with the large feature set already available in ALEGRA. A problem run with the multi-block structured physics option of ALEGRA runs in 1/2 the time and uses 1/5 as much memory as an unstructured version of the same problem. The multi-block structured mesh capability limits each block of mesh to have a regular i,j,k ordering of elements and nodes. However, the connections between blocks as well as the coordinates of nodes in blocks may be arbitrary. This flexibility allows description of a large set of geometries within the curvilinear multi-block limitations.

The approach used in developing the curvilinear multi-block capability was to add the underlying data storage and access routines to ALEGRA and

then enable as many of the existing ALEGRA capabilities as possible. A large portion of ALEGRA capabilities were inherited by the structured physics option with minimal effort. These included material libraries and input parsing. Another set of features required moderate effort to 'turn on' for structured physics: diatom input, programmed burn, artificial viscosity, input parsing, hourglass control, remeshing, and remapping. A third set of capabilities do not lend themselves to a transliteration style of implementation and require a structured mesh specific approach. The following capabilities are not yet available from the structured code: Adaptivity and Dynamic Load Balancing. The parallel decomposition and communication strategies of the structured mesh are significantly different from those of the unstructured mesh.

A significant difference apparent to a user of the structured mesh option relates to input and output. The presence of a '.gen' file is not assumed and the code does not automatically produce a '.exo' file. These files must be explicitly called out within the input file. This is because structured physics supports multiple input and output options.

Another significant difference is that for parallel runs the smallest unit of operations is a block of structured mesh. When a multi-block problem is run on more than one processor, the blocks are 'dealt out' to all of the available processors. If there are more processors than blocks, the calculation will progress but some processors will be essentially idle. Although a simulation run in serial may have more than one structured block, the least communications overhead is incurred if the problem consists of a single block. For parallel simulations a single block per processor will also limit communications overhead, however more blocks per processor will allow flexibility for upcoming dynamics load balancing capabilities.

Nomenclature

Table 1: Terms and Acronyms

Term	Definition
ACME	Algorithms for Contact in a Multiphysics Environment
ALE	Arbitrary Lagrangian Eulerian
ALEGRA	Arbitrary Lagrangian Eulerian General Research Application
NEVADA	Tool & code infrastructure for the ALEGRA family of applications
BC	Boundary Condition
EOS	Equation of State
IC	Initial Condition
MIG	Model Interface Guidelines
MMALE	Multi-Material Arbitrary Lagrangian Eulerian
SMALE	Single-Material Arbitrary Lagrangian Eulerian
MBS	Multi-Block Structured curvilinear
MPP	Massively Parallel Processor
SPMD	Single Program Multiple Data

1 Introduction to ALEGRA

ALEGRA [8, 37] is an ALE (Arbitrary Lagrangian-Eulerian) multi-material finite element code that emphasizes large deformations and strong shock physics. As an effort to combine the modeling features of modern Eulerian shock codes with the improved numerical accuracy of modern Lagrangian finite element codes, ALEGRA is a descendant of the PRONTO transient dynamics code [38, 39] and contains elements of the CTH family of shock wave codes [25, 7]. This capability permits a calculation to proceed in Lagrangian fashion until portions of the finite element mesh become too distorted, at which time the nodal points in the most deformed portion of the mesh are moved to reduce the distortion to acceptable levels. The advantage is that numerical dissipation is avoided until large deformations occur and this is limited to only those regions where severe distortions require mesh movement. In addition to mesh smoothing, the ALEGRA remesh algorithm can also move nodes to better resolve mesh regions with specific values of selected variables or their gradients.

ALEGRA is written predominantly in the C++ programming language [11], though we limit our use of some features of C++ to avoid efficiency problems. This allows us to take advantage of object-oriented programming techniques in managing the inherent complexity of the physics models being implemented. However, we also recognize the utility of incorporating various Fortran or C code and libraries if they best serve our modeling needs and if they are sufficiently mature and robust. Recent efforts in ALEGRA have focused on transforming the code into a "framework" for new simulation applications. To this end, most of the physics-specific treatments and references have been removed from the generic code base. The physics independent coding is known as NEVADA, to emphasize its complimentary nature to the another framework at SNL, SIERRA.

ALEGRA is designed to run on distributed-memory parallel computers using the Single Program Multiple Data (SPMD) paradigm, in which the mesh is decomposed into sub-meshes (i.e., domain decomposition). This is done because we need the enormous memories and processor speed of massively parallel processor (MPP) computers to analyze large, three-dimensional problems.

The unstructured mesh used by ALEGRA is taken from a GENESIS database which is the non-transient part of the EXODUS database [29]. The packages FASTQ [3], GEN3D [16], GJOIN [33], and CUBIT [36] can be used for preprocessing purposes to obtain an initial GENESIS mesh. The GENESIS file produced with FASTQ will be single precision only. CUBIT can generate double precision GENESIS files, but one can expect roundoff in the least significant figure of the nodal coordinates. The CUBIT tool accepts several common solid modeler output formats (*e.g.*, ACIS and IGES) as input, and then produces a finite element mesh in GENESIS format. In addition, there is an "exodus preference" available for Patran. The NEMESIS utilities based on Sandia's CHACO package [19] determine the decomposition of the mesh. Good mesh decomposition is important to minimize the memory requirements, balance the work on the compute nodes, and minimize the data passed between compute nodes. Whereas rectangular meshes are relatively easy to decompose, subdividing the arbitrary connectivity meshes used by ALEGRA is much more difficult. Recombination of the parallel result output files produced by ALEGRA can be done with the NEMESIS utility NEM_JOIN or with the CONCAT utility.

The mesh used by the MBS component of ALEGRA can be defined three ways: specified within the input file, imported as a modified PLOT3D file, or imported as a limited class of GENESIS files. The in-line mesh specification

allows definition of an `i,j,k` array of axis aligned blocks, each containing `l,m,n` uniformly distributed axis aligned elements. The modified PLOT3D file is generated by translation from a limited class of unstructured GENESIS files. Direct import of these GENESIS files is also available but should be limited to small problems (less than 100,000 elements). The element blocks in the appropriate GENESIS files must each consist solely of elements that can be re-ordered into an `l,m,n` array. There is no limitation on the ways that blocks may be connected to each other by contiguous nodes. The sidesets and nodesets are translated and imported along with the element block ids.

The plot output file from ALEGRA is in EXODUS database format [29]. The plot file can contain as many or few variables as the user desires and can include all the standard nodal and elemental variables, as well as material state variables. The EXODUS format output files may be post-processed using the BLOT [15], MUSTAFA (an internal Sandia visualization package), and Ensight (a commercial visualization package from Computational Engineering International, Inc.) graphics packages. The EXODUS files are platform independent.

In addition, ALEGRA provides both Eulerian and Lagrangian tracer particles that record time history data for selected variables (*e.g.*, pressure vs. time at the tracer location) in the HISPLT format, allowing post-processing with the HISPLT code [42]. The format of the history data file is binary and therefore a problem arises when the production platform is not binary compatible with the platform on which one desires to run HISPLT to examine the history data. In order to view the file with HISPLT in this case, the history data must be converted to text format, the file transferred to the second platform, and the data reconverted to binary format.

The restart output file from ALEGRA is in a separate format and cannot be used with any post-processing tools.

ALEGRA accepts mnemonic, free-format input. Descriptions of the keywords recognized by ALEGRA and their syntax requirements are provided later in this manual.

ALEGRA 2-D and 3-D versions are built separately. The 2-D versions handle both planar and axisymmetric geometries, permitting application to a reasonably broad class of problems. The 3-D version is restricted to Cartesian geometry.

ALEGRA is supported on many platforms on a routine basis. It can be

supported on any multiple processor compute platform which supports MPI message passing [17, 18] and has a relatively modern and robust compiler suite. ALEGRA does not utilize MPI-2 message passing features.

The ALEGRA release version numbers are on listed on CDs that are made available to select external users.

2 Overview

This chapter describes the use of the ALEGRA program, including mesh generation, problem specification, and post-processing. This chapter is arranged to take a user through the steps required to successfully run a simulation with ALEGRA. This section is not intended to describe in detail the physics or algorithms in ALEGRA. Several reference documents are available that cover these areas. Nor will this document provide all the detail necessary to perform the pre- and post-processing phases of an ALEGRA simulation. If you plan to use ALEGRA, you will want to acquire up-to-date manuals for the pre- and post-processing tools.

To successfully run a simulation with ALEGRA, one must 1) create a finite element mesh with tools like FASTQ, GEN3D, and/or CUBIT, or create a structured mesh using keywords inside the input deck in the next step. 2) create an ALEGRA problem specification input deck, 3) run ALEGRA, and 4) examine the results using analysis tools such as BLOT, MUSTAFA, ENSIGHT, and HISPLT. In addition, for parallel runs, one must learn how to use the `alegrabal` script and in some cases the `Concat` or `combinemp` scripts. This chapter is divided into the following sections:

- Basic ALEGRA Environment
- Running ALEGRA
- Example Problem
- Problem Reporting
- New Features

2.1 Basic ALEGRA Environment

The ALEGRA run time environment is based on:

- Environment variables
- Shell scripts
- Sandia's ACCESS system of finite element support tools

Anyone wanting to use ALEGRA needs to be acquainted with UNIX-based operating systems.

2.1.1 Environment Variables

The ALEGRA code relies on environment variables set within the user's `.cshrc` file. These environment variables are used within every context of the ALEGRA environment and allow code developers and users tremendous freedom in maintaining several versions of the ALEGRA code. At a minimum, the following environment variables must be set in order to run ALEGRA:

```
ALEGRA_ARCH
ALEGRA_EXE
ALEGRA_MIGDATA
```

`ALEGRA_ARCH` refers to an operating system or computer vendor along with a compiler description, if necessary. Acceptable values for `ALEGRA_ARCH` are: `linux-gcc`, `sun7`, `aix-5.1`, `irix64-6.5`, and `xtflop`. The `ALEGRA_ARCH` variable will be used to find a path within the Concat script, which is called by the Alegra script.

`ALEGRA_EXE` defines the path to the ALEGRA executable used by the Alegra script. Users will set this based on the version of ALEGRA they want to run.

`ALEGRA_MIGDATA` sets the path to the directory containing external files containing material data used by many material models. Use of this variable is described more fully in Chapter 12.

2.1.2 ACCESS Tools

Alegra assumes that some of the `ACCESS` tools are available in the user's path. [35]. The ACCESS system provides many pre- and post-processing tools for finite element codes.

2.1.3 Sandia Users Only

Sandia users (ESHPC lan, tflop lan, tflop-s lan) should establish the proper environment by sourcing the `alegra.users` script available in `$ALEGRA_ROOT/etc` on each machine. This script will define `ALEGRA_ROOT` on each platform according to the Table 2 and will define the appropriate value of `ALEGRA_ARCH`.

Table 2: Common `ALEGRA_ROOT` Paths

Platform	Path
ESHPC lan	`/pr/alegra`
janus	`/Net/projects/alegra`
edison	`/projects/alegra`
janus-s	`/projects/alegra`
pulsed power theory lan	`/apps/Alegra`

The `alegra.users` script will also modify the users `PATH` environment variable to include the appropriate directories.

2.2 Running ALEGRA

ALEGRA uses a common prefix for all of the files either read or written. This prefix is called the `runid`. The `runid` prefix can be any valid UNIX character string. The suffixes appended to `runid` determine the characteristics of the file. Figure 1 displays the possible files that are used in an ALEGRA simulation.

To use ALEGRA, a user must become proficient in the following procedures:

- Preprocessing or mesh creation
- User problem specification creation
- ALEGRA execution
- Post-processing or visualization

Throughout this process, users should consider where the natural boundaries among materials exist and how the materials should behave. In ALEGRA these boundaries define the boundaries of `BLOCK`s of elements. In the preprocessing phase, the element `BLOCK`s are given unique identifiers that can be tied to the problem specification input file. This coupling of identifiers allows the user to control physical and algorithmic parameters associated with the `BLOCK`. In addition, boundary conditions in the problem specification are tied back to unique identifiers in mesh creation that represent collections of nodes (`NODESET`s) and surfaces (`SIDESET`s). As users create a mesh, they will want to uniquely identify nodes and surfaces upon which to apply boundary conditions. Finally, some attention should be given to how `BLOCK`s and `MATERIAL`s will be used in the visualization process. For example, if a user wants to differentiate between two areas of the same `MATERIAL`, separate `BLOCK`s can be created and the `MATERIAL` duplicated in the problem specification.

2.2.1 Preprocessing

ALEGRA uses an unstructured finite element discretization for solving the governing partial differential equations. ALEGRA can begin with a body fitted mesh, a background mesh in which "shapes" are inserted, or a combination of both. The shape insertion option is useful for very complex geometries which can be accurately modeled in an Eulerian framework. This is the model that all Eulerian hydrodynamic codes use. ALEGRA accepts the same "shape" input (see the "`DIATOM`" specification in Section 5.7.1 on page 100) as the CTH code [7, 25].

The preprocessing steps for ALEGRA include:

1. Mesh Creation

2. Mesh Partitioning (Parallel Runs)

The creation of three-dimensional body-fitted meshes can be very time-consuming. Many years of research have gone into automating the meshing of solid model geometry. Many three-dimensional meshes require the user to decompose the geometry into simple parts that are essentially two-dimensional. These parts are then translated, warped, or rotated to make three-dimensional meshes. Great care must be taken to ensure that the

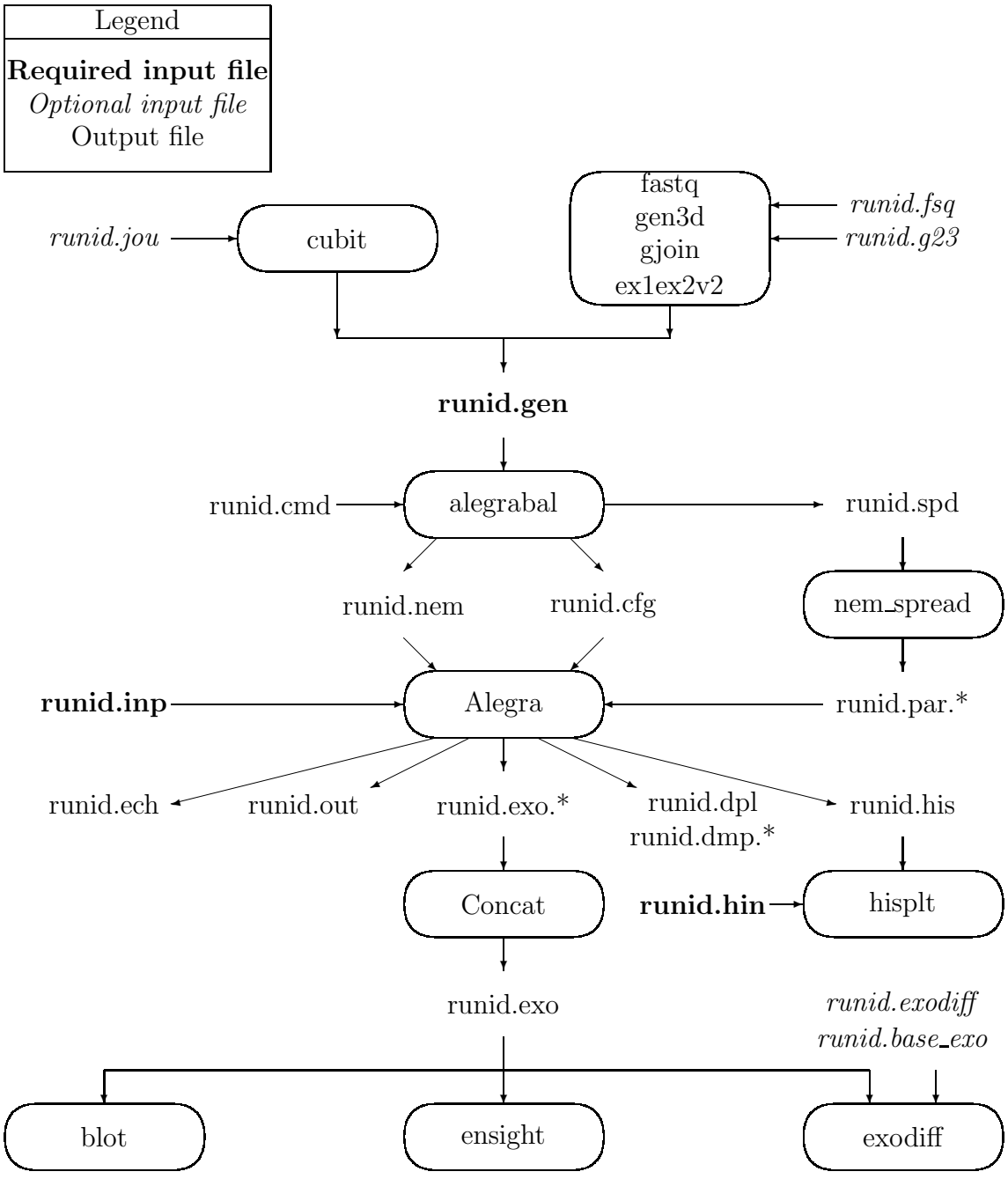

Figure 1: File names for ALEGRA.

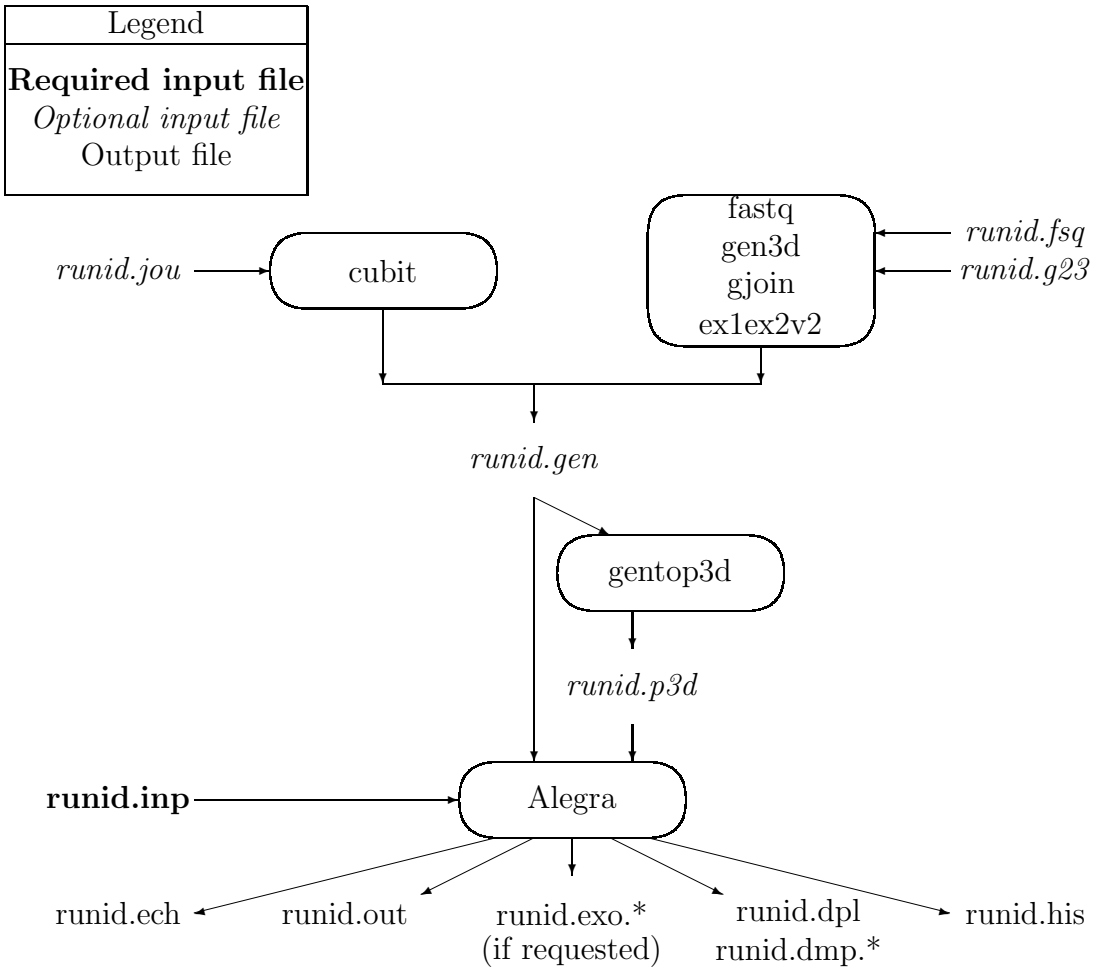

Figure 2: File names for ALEGRA MBS.

simple three-dimensional pieces can be "glued" together to form a coherent three-dimensional mesh.

There are two tools that are commonly used for mesh creation in ALEGRA: FASTQ/GEN3D, and CUBIT. In addition, the products of both tools can be combined with GJOIN to create an aggregate mesh.

FASTQ [3] is a two-dimensional meshing tool. Using this tool, a user creates point data, connects points to make lines and then connects lines to make blocks (regions) that can be meshed. The two-dimensional meshes can be rotated, translated and warped by GEN3D [16] to produce a three dimensional mesh. Three-dimensional meshes can be combined with GJOIN [33]. The majority of the time required to produce a complete three-dimensional mesh goes into assuring that the nodes of the three-dimensional sub-meshes match up at the connecting surfaces. Discontinuous mesh can be handled with the contact library [5] used in ALEGRA. The user must ensure that where contact is to be used, there are separate, unique nodes belonging to each side of the interface and that these nodes have not been merged in some fashion. All FASTQ generated meshes must also be processed by the utility EX1EX2V2 to transform the EXODUS I format to EXODUS II format. EXODUS I format is not supported by Alegra.

CUBIT [36] is an active mesh generation project at Sandia. The ultimate goal for CUBIT is to be able to take solid model geometry as input and provide a quality three-dimensional mesh based on user-defined tolerances. At present, CUBIT provides the functionality of FASTQ, GEN3D and GJOIN in a single package along with many enhancements and the ability to automatically mesh simple solid model geometries.

The GENESIS [29] file produced with FASTQ will be single precision only. CUBIT can generate double precision GENESIS files but one can expect round-off in the least significant figure of the nodal coordinates.

For execution in parallel, the mesh created by the preprocessing tools must be decomposed for the number of processors the run is to be executed on. This decomposition is performed using the `alegrabal` script, which in turn uses the NEMESIS [20] utilities. The minimum information required by the `alegrabal` script is the number of processors on which the run will take place and the `runid` of the problem. For example,

```
alegrabal -p 7 taylor
```

would break the `taylor.gen` mesh up for 7 processors for the default parallel platform. Additional options can be reviewed by running `alegrabal` with no parameter (or the `-h` parameter).

Figure 2 shows that for some structured grids computations it is possible to bypass the .gen file as structured grid information may be given in the .inp file.

2.2.2 Problem Specification File

ALEGRA uses a free-formatted ASCII input file to control the execution of the code. The units are assumed to be CGSK (cm-gm-sec-Kelvin). However, this can be changed with the `UNITS` keyword. Note that output from `TRACERS` to the HISPLT [42] database is always in SI units.

There is a one-to-one correspondence between the problem specification file and the mesh file. Integer identifiers used for `BLOCK`s, `NODESET`s, and `SIDESET`s can be used in the problem specification to attach attributes. Chapter 3 is devoted to this subject.

2.2.3 Executing alegra

ALEGRA is invoked using the `Alegra` script:

`Alegra` *runid*

where *runid* is a character string chosen by the user to identify the problem. This string is used to construct the names of the files that will be used for the calculation. For example, the command

```
Alegra taylor
```

will cause ALEGRA to look for a binary file named `taylor.gen` containing the problem mesh and a text file named `taylor.inp` containing the problem specification. Figures 1 and 2 depict the files ALEGRA will use for input and output. Table 3 describes the contents of each file. The files annotated with "Parallel" are only used when the code is run in multiprocessor mode.

Table 3: ALEGRA Input and Output Files.

File Name	Description
runid.inp	Problem specification. This file contains specifications on how long to run the problem, how many output dumps to make, what materials belong to the mesh blocks, how the mesh blocks behave and what physics package to run. Most of this document deals with the contents of this file. (Input)
runid.gen	GENESIS database. This file contains the description of the finite element mesh in EXODUS II format. A user can generate this mesh using various tools (FASTQ, GEN3D, CUBIT, etc.). This file contains the number of mesh blocks, the topology of the mesh, and the node and element sets that boundary conditions can be applied to. The mesh `BLOCK`, `SIDESET`, and `NODESET` IDs have a one-to-one correspondence with those used in the problem specification. (Input)
runid.nem	NEMESIS file. Describes to the NEMESIS `nem_spread` utility how the finite element mesh is to be decomposed onto N processors. This file is produced as a result of running the `alegrabal` script. (Input, Parallel)
runid.spd	Spread script. A script produced by the `alegrabal` script and run by the `Alegra` script to send portions of the .gen database to each of the N parallel processors, using the `nem_spread` utility. (Input, Parallel)
runid.cfg	Configuration file. A text file produced by the `alegrabal` script that is used by the NEMESIS utilities to both spread the files before the run and to combine the output files after the run. Its presence is a sign to the `Alegra` script that this is a parallel run. (Input, Parallel)
runid.cmd	Termination signal file. Using the Unix "`touch`" command to create a file of this name in the running directory will cause the code to terminate gracefully. (Input)

continued on next page

continued from previous page	
runid.nqs	Batch job file. As produced by `alegrabal`, this file is targeted to the NQS system on the Tflop machine. The script contains a commented batch system submission command that can be stripped out and entered on a command line. The command submits the script to the batch system, which then runs the `Alegra` script for this problem file. The `runid.nqs` file can be modified to be used on any batch execution environment. (Input, Parallel)
runid.ech	Problem specification echo. This file contains an echo of the problem specification. If errors occur in processing the problem specification, this file will point out the trouble spots. (Output)
runid.out	ASCII output file. This file contains a detailed list of the options set by the user and options set by the code. In addition, this file provides initial mass, momentum and energy associated with each mesh block. Finally, this file records when output is written to other files. (Output)
runid.exo	EXODUS database. This file contains both the GENESIS database and all element and node transient data requested by the user. By default, this file will contain nodal displacements and element volume fractions. Additional data must be requested in the problem specification file. (Output)
runid.his	HISPLT database. This file contains global, material and tracer location transient data. The HISPLT code can be used to extract this data and produce plots. (Output)
runid.dpl	List of restart files. This ASCII file is generated by ALEGRA and contains a list of all restart dumps that have been written for the problem. Upon restarting the code, ALEGRA searches this list for the restart dump specified in the input file, by dump number or time. (Output)
runid.dmp.*	Restart database. A restart dump file, together with the original GENESIS database and problem specification, contains all the information required to restart ALEGRA from a given time. However, this file does not work with any plotting packages. If an `EMIT RESTART` keyword is present, ALEGRA will by default retain only the last two restart dump files written.

continued on next page

continued from previous page	
	For serial runs, the dump files are named `runid.dmp.`n and `runid.dmp.`m, where $m > n$. `runid.dmp.`m will always be at the later time. The user can specify how many restart dump files to retain via the `RESTART DUMPS` keyword. These files can be very large. For parallel runs, the files are named `runid.dmp.`$N.n.m$, where N is the total number of processors used in the run, n is the the processor specific dump file (a number from 0 to $N-1$), and m is the dump sequence number described above. (Output) (Input, Restart)
runid.dbg	Debug file. With the correct user commands, extensive debugging information can be obtained and is written to this file. This information can be very useful to code developers, but is of little use to most users. (Output)

The `Alegra` script has the following options:

```
Alegra [-12] [-a] [-chBQ] [-C buffer_size] [-spF] [-b <batch queue>]
       [-I pfs_mode] [-H hostfilename] [-x executable] [-u mask]
       [-w warntime] [--migdata dirname] [-P procmode] runid

Options:
       1 - Mode 1 (default). Diagnostics written to screen
       2 - Mode 2. Diagnostics written to *.con.

       a - Use aprepro to preprocess the input file before running
           alegra

       c - Turn off crt control (allows running in background
           without hangup)
       h - Print this options message
       B - Beep to signal when calculation is complete
       Q - Quick look at user input correctness and mesh quality

       C buffer_size
         - For some architectures this changes the default buffer
           size

       s - run in serial
       p - run in parallel (appropriate files must exist)
```

```
F - Force rerun of nem_spread before running alegra

r num_raids   - number of raid arrays to use
--raid-offset - the raid offset

--nprocs size
  - run in parallel with 'size' processors
  - this implies a structured method

P - set processor mode for tflop
    0 = default; 1 proc/node; 3 = 2 proc/node;

S sub_dir - the sub directory
R root_dir - the root directory
f num_files - the number of concurrent files
I pfs_mode - mode for using the pfs (TFLOPs specific)

x executable
  - Use executable to perform the alegra calculation
u mask
  - set file creation mask
w warntime
  - have DPCS send SIGTERM to run warntime seconds before
    job time limit

b batch queue
  - batch queue to submit run to

H hostfilename
  - list of machine names for use in parallel runs
    (mpirun -machinefile hostfilename)

--migdata dirname
  - override the ALEGRA_MIGDATA environment variable
```

Generally speaking, the `p` and `s` options are not needed. The `Alegra` script along with the `runid.cfg` file created by the `alegrabal` script control how the ALEGRA executable is run.

Thus, the command line:

```
Alegra -2 -x /home/username/alegra.exe taylor
```

would use `alegra.exe` in `/home/username` and write the output to a file named `taylor.con`.

The APREPRO [32] utility is an algebraic preprocessor that reads a file containing both general text and algebraic, string, or conditional expressions. It interprets the expressions and outputs them to the output file along with the general text. The utility is part of the SEACAS [35] suite of pre and post processors. This capability allows the user great flexibility in writing input files that contain expressions that may depend on a few parameters and saves the effort manually changing values when performing parameter studies and the like.

In the process of running ALEGRA, many messages are printed to the console (UNIX "standard out"). There are five types of messages:

- Information
- Warning
- Fatal
- Global Record
- I/O Generation

Generally speaking, the information and warning messages that appear during the initialization of ALEGRA may be ignored. However, in the process of matching the problem specification deck to the mesh file a warning message can be generated that later causes a fatal message. For example, the code initially warns the user that a Sesame file could not be found if one does not exist. If the user requests a Sesame equation of state and a Sesame file could not be found, the code will reach a point of creating the Sesame equation of state object and the warning becomes a fatal message. When a fatal message is printed by ALEGRA, you should look through the warning messages to determine the cause.

Fatal messages cause the code to stop executing. There can be many causes for this. The user may need to interact with a developer or Sandia point of contact to determine the source of the error and if necessary submit a detailed problem report.

The global record message contains the cycle, time, time step, the element ID that limits the time step; the grind time (with I/O and without I/O); the

number of nodes remeshed since the last global record print; the total mass; and the total, internal and kinetic energies. The record also shows a count of the number of nodes, edges, faces, and elements in the problem, as well as the refine and unrefine count. The latter two values are related to "h-adaptivity", a research area of the code. The mass in the problem, mass loss and mass gain are also given. ALEGRA prints an explanation of the global record message at the start of each simulation. The frequency of printing a global record message is controlled with the `EMIT SCREEN` keyword. An example of this is shown below.

```
**************************************************************************
The following information will be dumped periodically to screen output

  (cycle_number)
  t=current_time
  dt=time_step
  (element_limiting_time_step)
  [grind_time,no_io_grind_time]
  {number_of_vertexes_moved}

  mass  = total mass
  mgain = mass gain
  mloss = mass loss

  te    = total energy
  ie    = internal energy
  ke    = kinetic energy
**************************************************************************

   ... lots of intervening output ...

(490) t=2.8422e-04 dt=5.9569e-07 (92) [3.6155e-05,1.7161e-05] {0}
 mass=7.3840e-02 mgain=0.0000e+00 mloss=0.0000e+00
 te=1.9143e+02 ie=1.3051e+02 ke=6.0919e+01

   ... lots more output ...
```

I/O generation messages inform the user of when the code sends information to the disk drives.

2.2.4 Post-processing

ALEGRA writes both EXODUS [29] and HISPLT [42] databases. The user can control the frequency at which the transient results are written to these files. The user can also control which variables are written to the EXODUS database. By default, mesh displacements and material volume fractions are written to EXODUS. HISPLT contains global, material and tracer particle information. The user can specify the number, type and location of tracer particles. Tracer points must be specified in the user input in order to have a valid HISPLT database. Variables included in the EXODUS and HISPLT databases can now be separately specified.

By default, ALEGRA writes single precision EXODUS files. This can be changed with the `DOUBLE PRECISION EXODUS` keyword described in Section 4.2.4 on page 77. The default is single precision because of the larger file sizes of double precision data. For parallel runs, use of the `Concat` tool will result in single precision combined EXODUS files. If the user desires to retain double precision files, then the `nem_join` tool must be used to do the concatenation step.

The EXODUS data base can be visualized by BLOT [15], MUSTAFA, and ENSIGHT. Most external users will use BLOT due to its portability. MUSTAFA is an SNL developed tool based on AVS Express and provides the most robust visualization environment for ALEGRA. ENSIGHT is a commercially available tool that supports input of EXODUS formatted files. It is currently used for ALEGRA's largest parallel simulations.

The HISPLT database is post-processed with the HISPLT code. Using the `runid.hin` file (user created input file) HISPLT creates a series of x-y plots that can be viewed by a number of device drivers. HISPLT and the device drivers are part of the CTH distribution.

2.3 Example Problem

This section presents an example of mesh construction, problem specification and graphical output of an ALEGRA run, the 3D Lagrangian Taylor Anvil Impact Problem. Numerous other examples are contained in the ALEGRA benchmark directories. These problems will be documented in the ALEGRA Verification document which is currently under development.

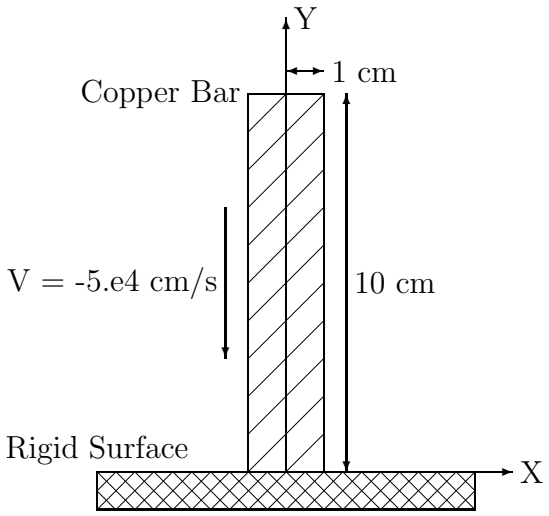

Figure 3: Taylor Anvil Simulation.

3D Lagrangian Taylor Anvil Impact Problem

The Taylor Anvil impact problem is a validation benchmark for both solid dynamics algorithms and constitutive models. In this problem, a solid cylindrical bar strikes a rigid target. Although this problem is two-dimensional axisymmetric, a quarter cylinder, three-dimensional mesh can be used to demonstrate the steps in using ALEGRA for three-dimensional Lagrangian calculations. The problem parameters are given in Figure 3.

The first step in running this problem requires building a mesh. For a three-dimensional mesh, this is a two-step process: 1) create a two-dimensional mesh and 2) rotate the two-dimensional mesh to create a three-dimensional mesh. Using FASTQ for the two-dimensional mesh, one can interactively input point data, create lines and regions, and then mesh the region. FASTQ can also read this data from a text file. For this problem, only four points are needed to create the two-dimensional mesh: (0, 0), (0,1), (1, 10), and (0, 10). These four points are then connected to form four lines. The four lines are connected to form a block. When creating the lines, the user will input the number of intervals desired on a line. In general, parallel lines will have the same number of intervals. In addition, the user will group lines to form boundary sets. For this problem the FASTQ input file, `taylor.fsq`, would look like:

```
POINT  1  0.00E+00  0.00E+00
POINT  2  1.00E+00  0.00E+00
```

```
POINT  3  1.00E+00  1.00E+01
POINT  4  0.00E+00  1.00E+01

LINE   1  STR  1  2  0   2  1.00
LINE   2  STR  2  3  0  20  1.00
LINE   3  STR  4  3  0   2  1.00
LINE   4  STR  1  4  0  20  1.00

REGION 1  1  -1  -2  -3  -4
SCHEME 0  M
BODY   1

ELEMBC 1  1       $ will become the Y = 0 plane
ELEMBC 2  2       $ will become the outer cylindrical surface
ELEMBC 3  3       $ will become the top surface
ELEMBC 4  4       $ will become the Y axis

NODEBC 1  1       $ will become the Y = 0 plane
NODEBC 2  2       $ will become the outer cylindrical surface
NODEBC 3  3       $ will become the top surface
NODEBC 4  4       $ will become the Y axis

EXIT
```

The FASTQ `REGION` corresponds to an ALEGRA `BLOCK`, the FASTQ `ELEMBC` corresponds to an ALEGRA `SIDESET`, and the FASTQ `NODEBC` corresponds to an ALEGRA `NODESET`. Note that there is no difficulty in numbering `SIDESET`s and `NODESET`s with the same ID as these are distinct entities. Graphically, this file is depicted in Figure 4. The meshed block is shown in Figure 5.

The following command generates a two-dimensional mesh file from the above `taylor.fsq` file:

```
fastq -m taylor.g2 taylor.fsq
```

The two-dimensional mesh is now stored in the file `taylor.g2`. The next step requires using this mesh as input to GEN3D. The following command begins the process:

```
gen3d taylor.g2 taylor.gen
```

Using the commands below, GEN3D generates a three-dimensional quarter cylinder with boundary sets on xy and yz planes and saves the resulting mesh

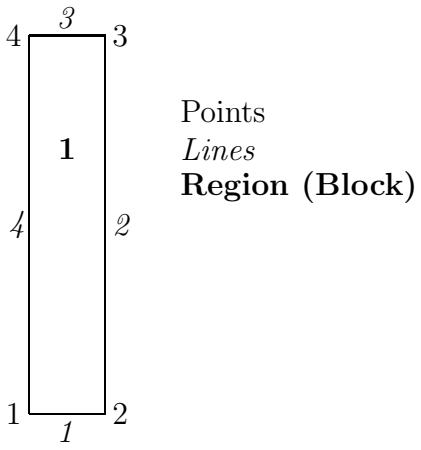

Figure 4: Point, Line and Block IDs for Taylor Anvil Problem.

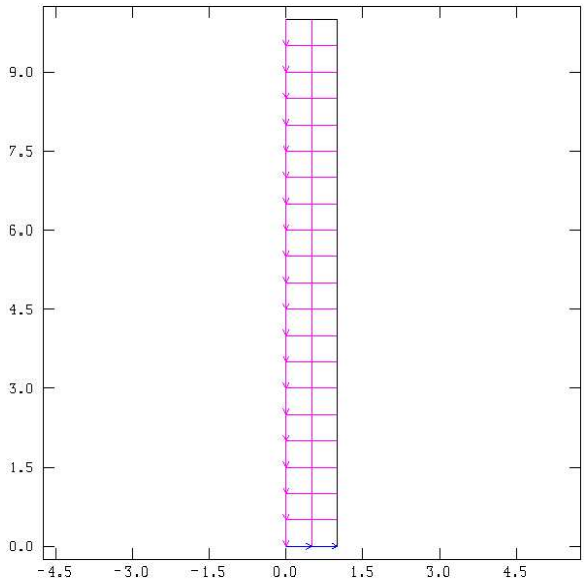

Figure 5: A Coarsely Meshed Taylor Anvil Problem.

in the file, `taylor.gen`.

```
rotate  4  90   $ rotate 90 degrees about the Y axis in 4 steps
ssets front 5   $ will become the Z = 0 plane
ssets back  6   $ will become the X = 0 plane
nsets front 5   $ will become the Z = 0 plane
nsets back  6   $ will become the X = 0 plane
exit
```

The FASTQ `SIDESET`s and `NODESET`s are rotated about the Y axis causing lines to become surfaces. Note however that `SIDESET` 4 and `NODESET` 4 become 5 distinct lines that are all collinear and cannot be visually distinguished in the GENESIS file. The GEN3D `SSETS` corresponds to an ALEGRA `SIDESET`, and the GEN3D `NSETS` corresponds to an ALEGRA `NODESET`. Again note that there is no difficulty in numbering `SIDESET`s and `NODESET`s with the same ID as these are distinct entities. These commands may also be entered into a data file, `taylor.g3d`, and used as input to GEN3D. The above command would then become:

```
gen3d taylor.g2 taylor.gen < taylor.g3d
```

The resulting mesh is shown in Figure 6.

With the mesh completed, the next step is to create the problem specification deck. Chapter 3 is devoted to describing the commands that can be used in running ALEGRA. In this problem, the user needs to describe the boundary conditions and how the material should behave. At this point one knows that:

1. The impacting material is copper. Therefore, it has strength and thus `SOLID DYNAMICS` is the correct physics. For this problem, an elastic plastic constitutive model and a Mie-Grüneisen Us-Up equation of state will suffice.
2. XZ, XY, and YZ planes cannot displace (IDs 1, 5, and 6)
3. The Y axis cannot displace, although its nodes can slide along the axis (ID 4)
4. Overlaying nodes along the Y axis should be constrained to remain overlaying

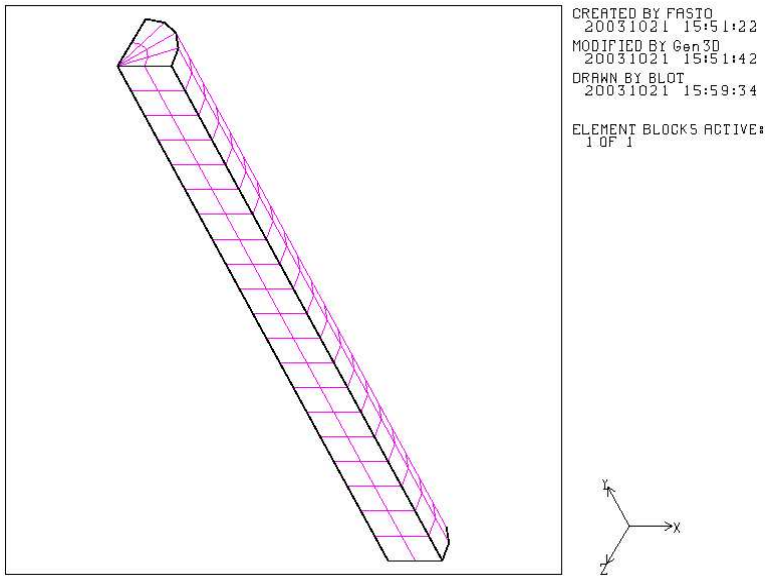

Figure 6: Three-dimensional Taylor Anvil Mesh.

5. Block 1 has an initial velocity of (0, -5.e4, 0)

6. Deformation probably will not be large and thus a Lagrangian mesh motion treatment will work. Selection of Lagrangian, ALE, or Eulerian mesh motion becomes more apparent with experience.

ALEGRA uses the concept of `DOMAIN`, `BLOCK`s, `MATERIAL`s, and material `MODEL`s. The `DOMAIN` describes attributes that apply to all of the `BLOCK`s. A `BLOCK` can have several `MATERIAL`s and has attributes that describe how the nodes that form the elements in the `BLOCK` will behave. Each `MATERIAL` can have several `MODEL`s. Boundary conditions and I/O control form other areas of the problem specification deck. The following example depicts an annotated problem specification deck for this problem.

```
title, "Taylor Impact"

termination cycle 200                       $ Job Control
emit screen, cycle interval 10              $ I/O Control
emit plot,   cycle interval 10

plot variables
```

```
  velocity
  density
  temperature
  pressure
end

solid dynamics                              $ Physics to be run
  no displacement, sideset 1, y             $ Rigid surface
  no displacement, sideset 4, x             $ Constrain the axis
  no displacement, sideset 4, z
  no displacement, sideset 5, z             $ Symmetry boundary
  no displacement, sideset 6, x             $ Symmetry boundary

  initial block velocity: block 1, y -5.0e4 $ Initial Condition

  degenerate bc: nodeset 4, axis,           $ Constrain nodes on the
                 x 0. y 0. z 0.             $ Y axis to move at COM
                 x 0. y 1. z 0.             $ velocity and acceleration

  domain                                    $ Default domain attributes
  end

  block 1                                   $ Block 1 will
    lagrangian mesh                         $ behave Lagrangian
    material 1                              $ and has Material 1
  end
end

material 1 "copper"             $ Material 1 has initial density
  density     8.932  $ gm/cm^3 $ and temperature. The stress
  temperature 398.   $ Kelvin  $ tensor will be evaluated with
  model 100                     $ constitutive model 100.
  model 2                       $ Pressure, temperature, and sound
end                             $ speed will be evaluated with model 2.

model 100 elastic plastic                   $ This constitutive model
  youngs modulus    1.076e+12 $ dyne/cm^2 $ is elastic plastic and
  poissons ratio    0.355                   $ populated with copper's
  yield stress      6.0e+9    $ dyne/cm^2 $ material parameters
  hardening modulus 2.0e+9    $ dyne/cm^2
  beta              0.5
end
```

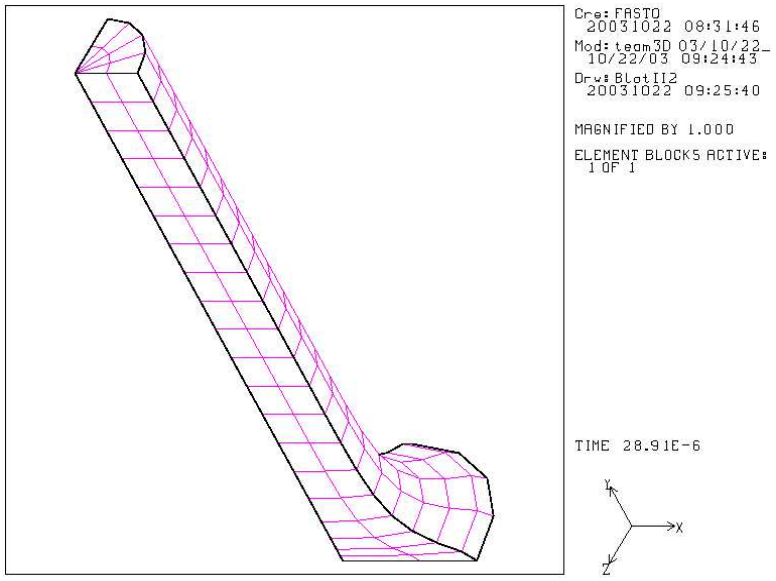

Figure 7: Taylor Anvil Deformed Mesh at 200 Cycles.

```
model 2 keos miegruneisen                $ This equation of state is
  matlabel = 'COPPER'                    $ that in the EOS_data
end                                      $ file for the material
                                         $ labeled 'COPPER'
exit
```

It is usually easiest to create a new problem specification by modifying an existing problem. The Taylor Anvil problem is run for 200 cycles using default parameters for time step control, artificial viscosity, and hourglass control. In the process of the simulation, plot dumps are added to `taylor.exo` every 20 time steps (cycles), and global information records are written to the screen every 10 cycles. In addition to the requested plot variables, the mesh displacements and material volume fractions are written to `taylor.exo`. This data can be viewed with BLOT or MUSTAFA. An image of the deformed mesh from the last plot dump is given in Figure 7.

To run this example with a structured mesh physics option several changes are required in the input deck. The `SOLID DYNAMICS` keyword changes to `STRUCTURED SOLID DYNAMICS`. The no displacement boundary must be applied to the corresponding nodesets instead of sidesets. The `DEGENERATE BC` must be commented out (not supported for structured). The GENESIS input

deck must be called out in the physics portion of the input deck as follows:

```
mesh, genesis
  file = "taylor.gen"
end
```

If exodus output is desired (the default is no plot output from structured physics options) then an output file name must be called out.

```
plot, exodus
  file = "taylor.exo"
end
```

2.4 Problem Reporting

The ALEGRA team uses standard web-based tool called SOURCEFORGE™ to track problems reported by users and as well as enhancement/feature requests. This tool is available to users and developers working within the SNL firewall. Generally users should work with a developer or other experienced user to ensure that their issue is a real problem. A bug or issue report can then be submitted along with attached files. A detailed description of the problem, an input file, files to generate the mesh (FASTQ, GEN3D, or CUBIT), and any output that might help diagnose the problem, should be included. The smaller the problem that is submitted illustrating the deficiency, the more likely that a fix will be found. Users in other locations should coordinate bug submission through their official Sandia contact.

2.5 New for Version 4.6

Version 4.5 was released in September 2003. Important new supported additions to the code include

1. Periodic wedges in unstructured meshes are supported provided at least two elements are attached to the wedge centerline.
2. Random density perturbation coding has been moved to a space-filling curve approach which is much more effective and gives the user some control over granularity.

3. User defined initial conditions allow C-language functions to be used to set initial values for physics variables using coordinates as input.

4. This release is the first to support the structured mesh capability.

3 General Input

ALEGRA input is divided into four general categories:

- execution control,
- physics specification,
- adaptivity control, and
- material modeling.

This section provides an overview of the input organizational structure and demonstrates the ways ALEGRA input can be organized with several examples. The ALEGRA input files should be organized into four sections for the four different categories of input. Any category can come first in the input file, but they can not be intermixed. A generic input file for an unstructured problem will look like the example shown below. Note that the adaptivity specification is an optional capability to control the H-adaptive feature of ALEGRA.

```
$ Execution Control Section
  Title
    A very general input file

  Output Control Keywords
  Other Control Keywords

$ Physics Specification Section
  Physics Specification Keyword
    $ Begin Physics specification subsections
    Region
      $ Region Keywords
    End

    Unstructured Region
      $ Unstructured Region Keywords
    End
```

```
    Mechanics
      $ Mechanics Keywords
    End

    Energetics
      $ Energetics Keywords
    End

    Dynamics
      $Dynamics Keywords
      End

    $ Other Physics subsections
    $ End of Physics subsection specifications
  End $ of the Physics Specification

$ Adaptivity Control Section
  Adaptivity Specification
    Enable Adaptivity
    $ Adaptivity control parameters
  End $ of the Adaptivity specification

$ Material Modeling Section
  Material 1
    Model 1
    Model 2
    $ Additional model specifications and variable
    $ initializations for Material 1
  End

  Model 1
   $ Model 1 Keywords
  End

  Model 2
   $ Model 2 Keywords
  End

  $ Additional Material and Model specifications
exit
```

In this very general input file, the ALEGRA keywords for each subsection (*i.e.*, level) of the physics specification are segregated into their own section. As we shall see later, this division is not necessary. It is done here to help explain the organization of this manual, which reflects the organization of the ALEGRA simulation object hierarchy. Sections 5.2 and 5.3 on pages 93 and 94 describe the legal physics specification keywords. Keywords available for the control of specific physics options are given in Sections 6 through 11. The order of entry of the various physics levels is of no importance. As described in Section 3.1.4 on page 63, the "$" is the comment character in ALEGRA input files. A specific example of an input file is shown below.

```
  TITLE
    SOD PROBLEM

  TERMINATION TIME 0.085
$ TERMINATION TIME = 0.5

  EMIT PLOT,   TIME = 0.005
  EMIT HISPLT, TIME = 0.005

  PLOT VARIABLES
    no underscores
    VELOCITY, as "VEL"
    PRESSURE: AVG
    DENSITY: AVG

  $ the following added to test min/max variable plots
    DENSITY  : MAX, AS "DENSI_MX" $ material scalar
    DENSITY  : MIN, AS "DENSI_MN"
    VOLUME                        $ element scalar
    VOLUME   : MAX, AS "VOLUM_MX"
    VOLUME   : MIN, AS "VOLUM_MN"
    MASS                          $ vertex scalar
    MASS     : MAX
    MASS     : MIN
    VELOCITY : MAX, AS "VEL_MX"   $ vertex vector
    VELOCITY : MIN, AS "VEL_MN"
  END

  SOLID DYNAMICS
```

```
  MECHANICS
    NO DISPLACEMENT, NODESET 1, Y
    NO DISPLACEMENT, NODESET 2, X
  END

  DYNAMICS
    PRONTO ARTIFICIAL VISCOSITY
      LINEAR = 0.15
      QUADRATIC = 1.2
    END
  END

  TRACER POINTS
    LAGRANGIAN TRACER 1 X= 0.795 Y= 0.1
    LAGRANGIAN TRACER 2 X= 0.895 Y= 0.1
    LAGRANGIAN TRACER 3 X= 0.995 Y= 0.1
    LAGRANGIAN TRACER 4 X= 1.005 Y= 0.1
    LAGRANGIAN TRACER 5 X= 1.105 Y= 0.1
    LAGRANGIAN TRACER 6 X= 1.205 Y= 0.1
  END

  BLOCK 1
    MATERIAL 1
  END

  BLOCK 2
    MATERIAL 2
  END
END

MATERIAL 1
  MODEL 1
END

MODEL 1 IDEAL GAS
  GAMMA   1.4
  RHO REF 1.0
  CV      2.066E7
  TREF    1.21E-7
END
```

```
  MATERIAL 2
    MODEL 2
  END

  MODEL 2 IDEAL GAS
    GAMMA   1.4
    RHO REF 0.125
    CV      2.066E7
    TREF    9.68E-8
  END

  crt: off
  EXIT
```

As a convenience to users, the segregation of physics specification control keywords into their subsections (`ENERGETICS`, `DYNAMICS`, etc.) is NOT required in the ALEGRA input file. The only physics keyword that must be present is that which describes the most specific characterization of the physics models, the options for which are found in Section 5 on page 92. An example of this feature is shown below, where the previous input file has been cast into this simpler format. Note that the `NO DISPLACEMENT` keywords are not placed within a `MECHANICS` keyword group, and that the `PRONTO ARTIFICIAL VISCOSITY` keyword group is not placed within a `DYNAMICS` keyword group, since in both cases, they already appear within the `SOLID DYNAMICS` keyword group.

```
  TITLE
    SOD PROBLEM

  TERMINATION TIME 0.085
$ TERMINATION TIME = 0.5

  EMIT PLOT,   TIME = 0.005
  EMIT HISPLT, TIME = 0.005

  PLOT VARIABLES
    no underscores
    VELOCITY, as "VEL"
```

```
  PRESSURE: AVG
  DENSITY: AVG

$ the following added to test min/max variable plots
  DENSITY  : MAX, AS "DENSI_MX" $ material scalar
  DENSITY  : MIN, AS "DENSI_MN"
  VOLUME                        $ element scalar
  VOLUME   : MAX, AS "VOLUM_MX"
  VOLUME   : MIN, AS "VOLUM_MN"
  MASS                          $ vertex scalar
  MASS     : MAX
  MASS     : MIN
  VELOCITY : MAX, AS "VEL_MX"   $ vertex vector
  VELOCITY : MIN, AS "VEL_MN"
END

SOLID DYNAMICS
  NO DISPLACEMENT, NODESET 1, Y
  NO DISPLACEMENT, NODESET 2, X

  PRONTO ARTIFICIAL VISCOSITY
    LINEAR = 0.15
    QUADRATIC = 1.2
  END

  TRACER POINTS
    LAGRANGIAN TRACER 1 X= 0.795 Y= 0.1
    LAGRANGIAN TRACER 2 X= 0.895 Y= 0.1
    LAGRANGIAN TRACER 3 X= 0.995 Y= 0.1
    LAGRANGIAN TRACER 4 X= 1.005 Y= 0.1
    LAGRANGIAN TRACER 5 X= 1.105 Y= 0.1
    LAGRANGIAN TRACER 6 X= 1.205 Y= 0.1
  END

  BLOCK 1
    MATERIAL 1
  END

  BLOCK 2
    MATERIAL 2
```

```
  END
END

MATERIAL 1
  MODEL 1
END

MODEL 1 IDEAL GAS
  GAMMA   1.4
  RHO REF 1.0
  CV      2.066E7
  TREF    1.21E-7
END

MATERIAL 2
  MODEL 2
END

MODEL 2 IDEAL GAS
  GAMMA   1.4
  RHO REF 0.125
  CV      2.066E7
  TREF    9.68E-8
END

crt: off
EXIT
```

3.1 Format and Syntax

ALEGRA takes as input a text file containing free format lines built around keywords or keyword groups. With few exceptions, the keywords or keyword groups may be in any order the user finds convenient.

3.1.1 Keywords

A keyword is a short sequence of English words denoting some action or quantity. For example,

```
TITLE
PISCES HOURGLASS CONTROL
VELOCITY VECTOR
```

are all examples of keywords.

The input routines are case insensitive and only enough characters of each word of a keyword need be entered to uniquely identify it. The number of words per keyword is significant and varies according to the specific keyword or keyword group. In the input syntax descriptions that follow, all keywords will be presented in `UPPER CASE`, while common grammatical constructs and numerical parameters whose values are supplied by the user will be shown in `lower case`. Optional keywords, constructs, or parameters will be enclosed in [square brackets]. Alternative choices for a keyword may be enclosed {curly braces} and separated by an `OR` symbol (|), as in {`ABC` | `DEF`}, meaning `ABC` or `DEF`.

New users may wish to start their input files with the keyword

```
DEBUG MODE: PARSER
```

which causes ALEGRA to print more extensive diagnostics of any errors it detects in the input.

3.1.2 Delimiters

Keywords may or may not require a numeric field or other grammatical construct to follow it (see for example Section 3.2 on page 63). *Adjacent keywords must be separated by a comma (,), colon (:), semicolon (;), equals sign (=), or newline; a blank is sufficient ONLY to separate a keyword from a numeric field, not one keyword from another.* (These delimiters may not always appear explicitly in the command descriptions that follow.) The user may optionally separate keywords and numeric fields using blanks, commas, colons, semicolons, equal signs, or newlines as seems appropriate. *The number of characters on an input line is limited to 160.* Placing more characters on a line can lead to platform-dependent results.

3.1.3 Keyword Groups

A keyword group is a sequence of keywords and numeric values bounded by a main keyword and the keyword `END`. All keywords within a keyword group may be in any order. For example:

```
PLOT VARIABLES
  PRESSURE, DENSITY, VONMISES
END

MODEL 4 LINEAR ELASTIC
  YOUNGS MODULUS = 2.71E8
  POISSONS RATIO = 0.25
END
```

3.1.4 Comments

Users may enter comments at any point by using a dollar sign ($) or asterisk (*). All text that follows a dollar sign on a line is ignored. Any line may be continued and lines may be combined. For example:

```
$ MATERIAL MODEL SPECS
  MODEL 1 ELASTIC PLASTIC $ ALUMINUM
```

3.2 Common Parameter Constructs

There are a number of common grammatical constructs that are used by more than one keyword. These constructs are described here, and they will be rendered in `lower case` in the descriptions of the keywords that use them. Users should replace the construct with the keywords described here. Remember that if a construct directly follows a keyword, an appropriate delimiter, *e.g.*, a comma (,) or equal sign (=), must separate them.

Other parameters that must be supplied by the user will also be presented in `lower case` in the input descriptions and should be replaced by the appropriate data type. Integer fields are specified by `int`, floating point fields are specified by `real`, and character strings are specified by `string` or some

other descriptive name. Default values for these parameters may be shown in parentheses following the descriptive name (*e.g.*, `real (2.5)` indicates that the default value of the real valued parameter is 2.5). All other options or sub-keywords will be discussed in the keyword descriptions and summarized in lists or tables.

3.2.1 block-id

```
BLOCK int
```

The GENESIS mesh file format used by ALEGRA assigns material numbers to subsets of the mesh. ALEGRA uses these material numbers to identify element blocks. The input specification file uses the `block-id` construct to identify these block numbers in keywords that specify material properties or initial conditions. For example:

```
INITIAL BLOCK VELOCITY, BLOCK 3, X = 20.0
```

The analyst must make sure that the proper block numbers are assigned during mesh generation. A reference to a nonexistent block number will cause ALEGRA to reject the input.

3.2.2 block-ids

```
BLOCK int [int] [int] ...
```

This is similar to the `block-id` keyword, except that any number of blocks may be specified.

3.2.3 nodeset

```
NODESET int
```

The `nodeset` construct specifies a nodeset from the GENESIS mesh file. It is used in keywords that apply to a set of nodes, such as a nodal boundary condition. For example:

```
NO DISPLACEMENT, NODESET 101, Y
```

The analyst must make sure that the proper nodesets are defined during mesh generation. A reference to a nonexistent nodeset will cause ALEGRA to reject the input.

3.2.4 sideset

```
SIDESET int
```

The `sideset` construct specifies a sideset from the GENESIS mesh file. This sideset can be used in keywords that must specify a set of edges (2D) or faces (3D), such as a traction boundary condition. For example:

```
PRESSURE BC, SIDESET 23, 0.25003
```

Sidesets are best thought of in terms of an element and associated face (3D) or edge (2D) so that an orientation for the sideset surface is implied. Sidesets interior to the mesh can be either single or double sided and correct usage depends on the particular application.

3.2.5 function-set

```
{ FUNCTION int [SCALE real] [SHIFT real] | real [SCALE real] }
```

The `function-set` construct specifies a user-defined `FUNCTION` table with an optional `SCALE` factor and `SHIFT`. If a keyword requires a `function-set` but only a constant value is needed, the `function-set` construct may take the form of a single `real` value. The `SCALE` option may be applied to this value.

ALEGRA makes extensive use of user-defined functions that are specified in the input specification file. (See the `FUNCTION` keyword in Section 5.13 on page 139.) These can be referenced one or more times by other keywords that require a functional specification of time or spatial dependencies. For example:

```
PRESCRIBED FORCE, NODESET 4, X, FUNCTION 1 SCALE 0.0005
```

Some keywords that use function sets support the SHIFT option. This option allows the user to specify the start time for the function data if it is different from the `START TIME` of the simulation. That is, the function is evaluated at time $t - t_{SHIFT}$. For example:

```
 FUNCTION 4 SHIFT 0.001
```

The function referenced by a keyword need not appear before the keyword, but must appear somewhere in the input specification file.

3.2.6 vector

```
X real [Y real [Z real]]            (Cartesian geometry)
R real [Z real]                   (Cylindrical geometry)
```

The `vector` construct specifies a 1-, 2- or 3-dimensional vector. The second form of the `vector` construct applies only to 2D cylindrical simulations. For example:

```
INITIAL VELOCITY, NODESET 10, X=0.300009 Y=0.033540 Z=0.699963
```

3.2.7 vector-function-set

```
X, function-set [Y, function-set [Z, function-set]]
```

The `vector-function-set` construct specifies a 1-, 2- or 3-dimensional vector function. Each component of the vector function is a separate ALEGRA `function-set` specification. For example:

```
X, FUNCTION 1, Y, FUNCTION 2, Z, FUNCTION 3
```

3.2.8 symtensor

```
XX real XY real XZ real YY real YZ real ZZ real       (3D)
XX real XY real YY real                               (2D)
```

The `symtensor` construct specifies a 2- or 3-dimensional symmetric tensor. The first form should be used for 3D and the second form for 2D. For example, in 3D:

```
TRACTION BC, SIDESET 3,
   XX 0.0  XY 1.0  XZ 0.0
           YY 0.0  YZ 0.0
                   ZZ 0.0
```

3.2.9 direction-function

```
{{ X | Y | Z | R }, function-set |
 { RADIAL | NORMAL | TANGENT }, function-set,
 vector, [CENTER, vector] }
```

The `direction-function` construct combines a direction with a `FUNCTION` that is to be applied. In the first form, a direction along one of the primary axes is specified, namely `X`, `Y` or `Z` in Cartesian geometry or `R` or `Z` in 2D cylindrical geometry. This is followed by a `function-set`. For example:

```
PRESCRIBED FORCE, NODESET 4, X, FUNCTION 1 SCALE 0.0005
```

In the second form a direction relative to a reference point is specified, namely `RADIAL`, `NORMAL` or `TANGENT`. This is followed by a `function-set` and a `vector`. Lastly comes the reference point as specified by the optional `CENTER` keyword. It is assumed to be the origin if omitted. For example:

```
PRESCRIBED VELOCITY, RADIAL, FUNCTION 2, X 1.0 Y 1.0,
                     CENTER, X 0.0 Y 0.0
```

Note that the `vector` input must be provided with the `RADIAL` option even though it will not be used. The `CENTER` option provides for user specification of a center for the `RADIAL` computation other than the origin.

3.2.10 time-or-cycle-interval

```
{ [EXACT] TIME [INTERVAL] real | CYCLE [INTERVAL] int }
```

The `time-or-cycle-interval` construct specifies an interval of time or interval of cycles used by the keyword. Note that time intervals are specified as floating point values and cycle intervals are specified as integer values.

The extra word `EXACT` is optional for the `TIME INTERVAL`. If specified ALEGRA will modify the time step so that output is emitted at the exact interval. Otherwise the default behavior of ALEGRA is to emit output when a time interval is matched or exceeded. The intent of the `EXACT` modifier is to synchronize long running verification simulations on various computer platforms. `EXACT` is available for the `EMIT PLOT`, `EMIT HISPLT`, and `EMIT RESTART` commands.

The extra word `INTERVAL` is optional and may be included for readability. For example:

```
EMIT PLOT,   TIME  INTERVAL 0.1
EMIT OUTPUT, CYCLE INTERVAL 25
```

3.2.11 time-range

```
FROM [TIME] real TO real
```

The `time-range` construct specifies a range of time values for which a keyword will apply. The extra word `TIME` is optional and may be included for readability. For example:

```
EMIT PLOT: CYCLE INTERVAL 1, FROM TIME 0.0 TO 0.3e-6
```

4 Execution Control

4.1 Job Initiation and Termination

4.1.1 Title

```
TITLE
  string
```

The line of input text following a `TITLE` keyword is used as a title string for the problem. Currently, only eighty characters are recorded. For example:

```
TITLE
  CASE 4 WITH 40,000 ELEMENTS
```

4.1.2 Exit

```
EXIT
```

The `EXIT` keyword signals the end of processing for an input specification file. ALEGRA will ignore the remaining contents of the file. This allows an extended explanation of the problem to be placed at the end of the input file (a highly recommended practice).

4.1.3 Units

```
UNITS, {CGS | SI}
```

Default units for ALEGRA are `CGS` units (cm, g, sec, K). However, the `UNITS` keyword may be used to change the default units to `SI` (System International, m, kg, sec, K). Default temperatures are always Kelvin. An example:

```
UNITS, SI
```

Units for individual numerical (float) quantities can be entered in the input immediately following the value. This functionality is applicable for all keywords that accept a floating point argument except those in the `DIATOM` input section (Section 5.7.1 on page 100). The units specified will be used to convert the float value to the run units according to a set of basic fundamental unit exponents (length, mass, time, and temperature). The units labels and definitions that can currently be parsed by ALEGRA are shown in Table 4. The unit label designation of `eV` (temperature associated with electron volts) are included for compatibility with CTH input variables. *Electron volts require conversion to Kelvin for either the `SI` or `CGS` systems.*

To be in a readable format, the units string must be entered with square brackets [ ]. The following operators are recognized: parentheses ( ) for multiplication, the forward slash (/) for division, and the carat (^) for exponentiation. Units are not required for any keyword in the input file, but are merely intended as a user convenience. Acceptable unit designations are listed in Table 4.

Examples:

```
R0 2730.  [kg/m^3]
CV 1.4e11 [erg/((gm)(eV))]
cv 1.1e11 [erg/gm/ev]
```

4.1.4 Read Restart

```
{ READ RESTART TIME real | READ RESTART DUMP int }
```

This keyword specifies which restart file is read to obtain the initial values for all variables at restart. The first form provides the time, a `real` value that may be positive or negative. The second form specifies the number of the actual `RESTART DUMP` to be read.

ALEGRA generates restart files with names of the form `runid.dmp.n` at time intervals specified by the `EMIT RESTART` keyword, where `n` is an integer, starting with 0. The number of dumps that are retained can be controlled through the `RESTART DUMPS` keyword. By default only the last two files generated will be retained. Earlier restart dumps are deleted as the calculation

Table 4: Allowable Unit Designators

Full Name	Unit System Association	Allowable Designator	Fundamental Unit L = length M = mass t = time T = temperature
meter	SI	m	L
kilogram	SI	kg	M
gram	CGS	gm	M
second	SI, CGS	s	t
Kelvin	SI, CGS	K	T
centimeter	CGS	cm	L
Electron volt	CGSEV	eV	T
erg	CGS	erg	ML^2/t^2
dyne	CGS	dyn	ML/t^2
Newton	SI	N	ML/t^2
Joule	SI	J	ML^2/t^2
Pascal	SI	Pa	$M/(t^2\ L)$
Watt	SI	W	ML^2/t^3

progresses to conserve disk space. For parallel runs, dump files are written for each processor and are named `runid.dmp.N.m.n`. Here `N` is the total number of processors, `m` is the number of an individual processor and `n` is the dump number, as above.

A dump list file named `runid.dpl` is generated that contains a list of all restart dumps that have been written for the problem. ALEGRA searches this list for the desired restart dump. The restart dump file to be read will be the one whose time is closest to the specified time, if the restart dump number is not specified directly. Thus one can specify a restart time that is close to the time at which the restart dump was written and that dump will be selected.

If a restart time greater than that of the last restart dump is specified, the last dump will be read. If a restart time less that the problem `START TIME` is specified, the last dump will be read provided the dump list file exists and contains a list. Otherwise, the calculation will begin as a new one starting at the initial time.

No restart dump is used if this keyword does not appear. Instead, a new

calculation is started with all quantities set to initial values calculated from the input specification file.

Examples:

```
READ RESTART TIME 26.5E-6
READ RESTART DUMP 12
```

Some output files (*e.g.*, `runid.out` and `runid.his`) generated from a restart will have an `_n` appended to their name, where `n` is selected to be unique in the directory in which the files are written, unless the `OVERWRITE FILES` option is chosen.

The `READ RESTART DUMP` keyword allows a negative dump number. Specification of a negative dump number causes ALEGRA to check for a `*.dpl` restart file. If this file does not exist or is empty, then the simulation will begin as a new simulation. Otherwise, the simulation will restart at the latest available restart dump. This behavior is similar to specifying a restart time prior to the simulation start time on the `READ RESTART TIME` command.

```
READ RESTART DUMP = -1
```

4.1.5 Start Time

```
START TIME real
```

This keyword specifies the starting time for the problem if it is different from zero. The `real` value specified for the `START TIME` may be positive or negative. This option is useful if the simulation is driven by experimental data that may have a non-zero `START TIME`.

4.1.6 Termination CPU

```
TERMINATION CPU real
```

This keyword specifies the maximum cpu time, measured in seconds, at which the problem calculation is to terminate. If any processor running a

parallel ALEGRA calculation exceeds this limit then the whole calculation will shut down gracefully. Generally this option is used when running in batch mode. The `TERMINATION CPU` value should be set to a value smaller than the batch request time limit in order to allow the problem to terminate before it is killed by the system batch manager. The user must determine this value for a given run depending on the number of files that will be written at the end of the calculation and the expected cycle time since there is no cycle time estimation capability in the code.

The `TERMINATION CPU` keyword must be used in conjunction with either the `TERMINATION TIME` and/or a `TERMINATION CYCLE` keyword. `TERMINATION CPU` by itself is not sufficient and the problem will not run. If more than one of `TERMINATION CPU`, `TERMINATION TIME`, and `TERMINATION CYCLE` are specified, the problem will terminate when any of the criteria are satisfied. A restart record will be written when this option ends the run if any type of `EMIT RESTART` command has been issued. A history record and an EXODUS dump will also be written if these have been specified with `EMIT` keywords. The termination of the run due to cpu limit will be noted in the user's terminal window (*i.e.*, the "standard output" device/file).

4.1.7 Termination Cycle

```
TERMINATION CYCLE int
```

This keyword specifies the cycle on which the problem calculation is to terminate.

If more than one of `TERMINATION CPU`, `TERMINATION TIME`, and `TERMINATION CYCLE` are specified, the problem will terminate when any of the criteria are satisfied. This form of termination control can be used by itself. A restart record will be written when this option terminates the run. A history record and an EXODUS dump will also be written if these have been specified with `EMIT` keywords. Termination due to cycle limit will not be reflected in the user's terminal window - the code will simply stop.

4.1.8 Termination Time

```
[EXACT] TERMINATION TIME real
```

This keyword specifies the problem time at which the problem calculation is to terminate. By default, ALEGRA may overshoot the `TERMINATION TIME` by some fraction of a time step. If the optional prefix `EXACT` is added to this command, then the time steps for the last ten cycles will be adjusted as necessary to ensure that the calculation terminates at the exact time specified.

If more than one of `TERMINATION CPU`, `TERMINATION TIME`, and `TERMINATION CYCLE` are specified, the problem will terminate when any of the criteria are satisfied. This form of termination control can be used by itself. A restart record will be written when this option terminates the run. A history record and an EXODUS dump will also be written if these have been specified with `EMIT` keywords. Termination due to time limit will not be reflected in the user's terminal window (*i.e.*, the "standard output" device/file) - the code will simply stop.

4.2 I/O Control

4.2.1 Copy Input

```
COPY INPUT, { TRUE | FALSE | ON | OFF }
```

By default, ALEGRA will copy the entire contents of the input file to the information data records of the EXODUS [29] output file. For very large input files, *e.g.*, those having long `FUNCTION` definitions or large numbers of `DIATOM PACKAGE`s, this can be slow and tedious and result in long initialization times. The default behavior can be suppressed and initialization times can be shortened by setting `COPY INPUT` to `FALSE` or `OFF`. This capability is only enabled for unstructured grids.

4.2.2 CRT

```
CRT, { ON | OFF }
```

This keyword enables or disables the function which checks the keyboard buffer for any keystrokes. This method allows for termination of the calculation at the end of the current cycle when the user enters the word `STOP` into

the keyboard buffer. Executive and query menus are also accessed by this method when the user enters the word HELLO into the keyboard buffer. The CRT is ON by default. This precludes straightforward background running of a calculation, however, so to run a calculation in the background, the user should turn the CRT function OFF, *i.e.*, type a CRT, OFF line into the input file.

4.2.3 Debug Mode

```
DEBUG MODE: { ADAPT (OR ADAPTIVITY) | COMM STATS | CONNECTIVITY |
              SUMP PROCSET | EVATTR | EXODUS | FILE |
              GENIIFILEOPEN | LOCATION | PARSER | PERIODIC BC |
              PROCSET | PROFILETIME | PROFILEMEMORY |
              PROFILEHARDWARE |REMAP | RESTART | STORAGE |
              TIMESTEP | TRACKER }
```

This keyword enables debugging information output. These options are generally used by developers for diagnostic purposes and are provided with only limited user support. More than one flag may be turned on by repeating the DEBUG MODE keyword.

- ADAPT activates h-adaptivity diagnostics.
- COMM STATS will display information about the number and types of interprocessor communication. This display will take place at the end of the run.
- CONNECTIVITY turns on comprehensive connectivity checking of the topology database (vertex, edge, face, and element connectivity). If an invalid database is found the calculation is terminated.
- DUMP PROCSET forces extensive output dumps of processor set related information.
- EVATTR outputs information on element and vertex attributes.
- EXODUS turns on EXODUS II [29] warnings. (This option will cause the EXODUS library flag EX_VERBOSE to be turned on, leading to messages being printed from these EXODUS library routines.)

- `FILE` sends information to the `runid.dbg` output file on all processors and is used in conjunction with other debugging flags. Also turns on debug outputs associated with the opening of plot output files with the adaptivity option active.
- `GENIIFILEOPEN` sends an informative message when each genesis II file is opened. This is useful when trying to debug and track the load process on large numbers of processors.
- `LOCATION` gives information on the current program step location.
- `PARSER` turns on verbose mode in the parser. This will result in more information being delivered when errors occur in the parsing of the input. For example, the entire sequence of valid keywords will be written to the output following the error message, or suggestions about possible causes of the error will be printed.
- `PERIODIC BC` prints diagnostic information about nodesets used in the periodic boundary conditions
- `PROCSET` gives information on the processor sets.
- `PROFILETIME, PROFILEMEMORY, PROFILEHARDWARE` turns on a collection of performance information and outputs to the runid.out file. `PROFILETIME` and `PROFILEMEMORY` are supported on all platforms and give timing and memory usage. `PROFILEHARDWARE` is supported only on some platforms and gives information such as floating point operation rates. The collected data may be machine specific, and on platforms that do not support the collection of hardware performance counters, there is no output. Additional information on performance analysis can be found in Section 14.
- `REMAP` gives information on the remap step.
- `RESTART` activates diagnostics for debugging restarts.
- `STORAGE` turns on memory usage information to output to the .out file
- `TIMESTEP` gives time step control information.
- `TRACKER` gives information on the interface tracker.

For example,

```
DEBUG MODE: LOCATION
DEBUG MODE: PARSER
DEBUG MODE: EXODUS
```

4.2.4 Double Precision Exodus

```
DOUBLE PRECISION EXODUS
```

This keyword indicates that the GENESIS [29] input file uses double precision for floating point numbers. The default is for the GENESIS file to use single precision for floating point numbers. The GENESIS file produced with FASTQ [3] will be single precision only (although the FASTQ mesh must be converted to an `EXODUS VERSION TWO` file before it is used by ALEGRA). CUBIT [36] can generate double precision GENESIS files, but one can expect roundoff in the least significant figure of the nodal coordinates. Both single precision data and slightly noisy double precision data will affect results if the simulation is sensitive to small perturbations in face orientation.

This keyword also indicates that the EXODUS [29] output file is to use double precision for floating point numbers. The ALEGRA default is to produce single precision files because of the larger file sizes of double precision data. For parallel runs, use of the `Concat` tool will result in single precision combined EXODUS files. If the user desires to retain double precision files, then the `nem_join` tool must be used to do the concatenation step.

4.2.5 Exodus Version

```
EXODUS VERSION TWO
```

`EXODUS VERSION TWO` indicates that the GENESIS input file and the EXODUS output file are in EXODUS II format [29]. This is a deprecated keyword because only `EXODUS VERSION TWO` GENESIS files are currently read by ALEGRA. The keyword is ignored in the input stream.

4.2.6 Emit Hisplt

```
EMIT HISPLT, time-or-cycle-interval [time-range]
```

This keyword causes ALEGRA to emit a HISPLT [42] record to the `runid.his` file at specified intervals over an optionally specified time range. If no time range is specified, the records are emitted throughout the run at the specified time or cycle interval. The user can have multiple entries of this card, thereby creating a time table. The input specification can take one of three forms, depending on the form of the `time-or-cycle-interval` construct and whether the `time-range` construct is specified. For example:

```
EMIT HISPLT, TIME INTERVAL = 0.0001
EMIT HISPLT, TIME INTERVAL = 0.0003, FROM 0.001 TO 0.002
EMIT HISPLT, CYCLE INTERVAL = 20
```

For the second example above, information will be written starting at time=0.001 and at times separated from 0.001 by 0.0003, but information will not be written for time=0.002. Thus, this example would produce HISPLT dumps at 0.001, 0.0013, 0.0016, and 0.0019. Several specifications of the second type can be used to setup a series of outputs at varying time intervals throughout the problem run. For the first and third specifications, however, only the first occurrence is used to set the interval of outputs for the entire run.

HISPLT has limitations of no more than 20 materials in a calculation and no more than 1002 tracer points [42].

4.2.7 Emit Output

```
EMIT OUTPUT, time-or-cycle-interval [time-range]
```

This keyword causes ALEGRA to emit various summary information to the `runid.out` file.

4.2.8 Emit Plot

```
EMIT PLOT, time-or-cycle-interval [time-range]
```

This keyword causes ALEGRA to emit an EXODUS [29] plot record at specified intervals over an optionally specified time range. If no time range

is specified, the records are emitted throughout the run at the specified time or cycle interval. As explained in the `EMIT HISPLT` keyword (Section 4.2.6 on page 77), the user can have multiple entries of this card, thereby creating a time table. EXODUS plot records are written in one of two possible forms.

For an unchanging initial topology, such as topologies produced with no adaptivity, or initial mesh refinement, a traditional EXODUS file is produced: a GENESIS mesh database with subsequent time slice information appended for each plot event. The single plot file has the name `runid.exo`.

For adaptive problems and conditions where the topology is not guaranteed to be constant, then individual EXODUS plot records are written to separate files. The current GENESIS mesh database corresponding to the EXODUS output is contained in each of these files. The multiple plot files have the names `runid.hat.n` where `n` corresponds to the $\mathtt{n}^{th}$ plot dump.

4.2.9 Emit Restart

```
EMIT RESTART, time-or-cycle-interval [time-range]
```

This keyword causes ALEGRA to emit a restart dump at specified intervals over an optionally specified time range. If no time range is specified, the dumps are emitted throughout the run at the specified time or cycle interval. The user can have multiple entries of this card, thereby creating a time table, as explained in the `EMIT HISPLT` description (Section 4.2.6 on page 77).

Restart dumps permit a user to break a single calculation into several runs. They also help to avoid losing everything if the computer crashes in the middle of a run.

If `EMIT RESTART` is specified and the calculation terminates before the specified time or cycle, a restart dump is written at the termination time or cycle. (If an `EMIT RESTART` option is not specified, no restart dumps will be written.)

The dump files will be named `runid.dmp.n`, where `runid` is the prefix of the ALEGRA input specification file, *i.e.*, the file name used on the ALEGRA execute line, and `n` is a dump number. The dump number begins with 0 for the first restart dump and is incremented by 1 each time a restart dump is written. By default, only the two most recent restart dump files are retained;

earlier dump files are deleted as the calculation progresses to conserve disk space. However, the number of files retained can be controlled with the `RESTART DUMPS` keyword. A list of the times of all restart dumps is written to the file `runid.dpl`, which is then read when restarting to determine the proper dump file from the restart time specified.

For parallel runs, dump files are written for each processor and are named `runid.dmp.N.m.n`. Here `N` is the total number of processors, `m` is the number of an individual processor and `n` is the dump number, as above.

Restart dumps are operational with the `INITIAL REFINEMENT` option specified in the `DOMAIN` keyword group (see Section 5.10.3 on page 132).

4.2.10 Emit Screen

```
EMIT SCREEN, time-or-cycle-interval [time-range]
```

This keyword causes ALEGRA to emit a short summary of the state of the calculation at specified intervals over an optionally specified time range to the users terminal (*i.e.*, the "standard output" device/file). If no time range is specified, output is emitted throughout the run at the specified time or cycle interval. The user can have multiple entries of this card, thereby creating a time table, as explained in the `EMIT HISPLT` description (Section 4.2.6 on page 77). The default screen output frequency is every 10 cycles. An example of the global record message produced by this keyword is shown in Section 2.2.3 on page 44.

4.2.11 Overwrite Files

```
OVERWRITE FILES
```

This keyword causes output files from a preceding run to be overwritten on a restart. The default action is to not overwrite the plot and history files, instead appending an "`_n`" to an output file name upon a restart, where `n` is an integer 0, 1, 2, ... selected to be unique in the directory in which files are being written.

4.2.12 Plot Variables

```
PLOT VARIABLES
   [NO DEFAULT OUTPUT]
   [NO REGION VARIABLES | ALL REGION VARIABLES]
   [NO MATERIAL GLOBALS | ALL MATERIAL GLOBALS]
   [NO UNDERSCORES]
   name [conversion]
   name [conversion]
   ...
END
```

This keyword permits the user to specify the variables to be written to the EXODUS [29] plot file. If a variable name is repeated in the `PLOT VARIABLES` keyword group, then it will be written to the database more than once. By default, `VOLUME FRACTION` and the node displacements, `DISPLX`, `DISPLY`, and `DISPLZ`, are the only variables written to the database, along with all global and material global variables. If no default output is desired, the keyword `NO DEFAULT OUTPUT` should be entered. The variables listed after this keyword will be the only variables listed to the file. The writing of all global variables and/or all material global variables can be switched off or on by entering the keywords `NO REGION VARIABLES`, `ALL REGION VARIABLES`, `NO MATERIAL GLOBALS`, or `ALL MATERIAL GLOBALS`.

With the exception of global variables, the name of any registered variable can be used as a value for `name`. Registration of variables is dependent upon which physics and material models are requested by the user in the input specification. The list of registered variables for any given problem can be found in the `runid.out` file produced by a run of ALEGRA. Global variables, which can not be used as plot variable names, are listed as `REGION_VAR`s in the `runid.out` file.

For convenience, some more common names are given in the following tables. Those variables marked with an asterisk (*) are not normally stored for the entire grid; inclusion of these in the data base causes ALEGRA to allocate additional dynamic memory for their storage. The `MATERIAL` variables available for plotting are very dependent upon the particular material `MODEL`s, so only the most generic are listed. The names of the variables available for plotting in each material model are given in the tables in Section 12 on page 197. The variable `types` are also given in the tables to aid in the

proper choice of conversions, if any.

In the output EXODUS files, the variable names will appear and be used to specify what data is available for display by various post-processors. The EXODUS variable names will be the same as the names included in the `PLOT VARIABLES` keyword group, except that blanks within a name will be replaced by underscores. An exception to this rule can be forced by the user (see `NO UNDERSCORES` below). The actual names of variables that have been written to the EXODUS file can be found by using the ACCESS [35] GROPE [34] utility and the `LIST NAMES` command.

For `vector` variables, such as `VELOCITY`, each vector component will be written to the EXODUS file with the name of the component direction appended. For example, the `x` component of velocity will be named `VELOCITY_X`.

For `material` variables, the EXODUS name of the variable describing the value of the quantity for a given material is indicated by appending a "`_N`", where `N` is the material id number assigned by the user in the input file with the `MATERIAL` keyword. For example, the temperature of the material that is labeled 101 by the user is written to the EXODUS file with the name `TEMPERATURE_101`.

`tensor` quantities are labeled similarly, such as `ARTIFICIAL_VISCOSITY_XY` for the `xy` component of the artificial viscosity tensor in an element. `tensor` quantities that are specific to a `material` will be named with a "`_N`" appended to the variable name before the direction designation. For example, `STRESS_202_XX` will be the name of the `xx` component of stress in material 202 in an element.

Valid values for the optional `conversion` keyword are: `MAGNITUDE`, `VOLUME AVERAGE (AVG)`, `MASS AVERAGE`, `MAXIMUM`, and `MINIMUM`.

The `MAGNITUDE` takes the absolute value of a scalar. The `MAGNITUDE` keyword causes the magnitude of other quantities such `vector`, `symtensor` or `tensor` variables to be dumped instead of the individual components. The MAGNITUDE of the vector $\mathbf{x}$ is defined by the Euclidean vector norm $||\mathbf{x}||_E = \sqrt{\mathbf{x}_i^2} = \sqrt{\mathbf{x}^T\mathbf{x}})$. The MAGNITUDE of the tensor $\mathbf{S}$ is the Euclidean matrix or Frobenius norm defined by $||\mathbf{S}||_E = \sqrt{\mathbf{S}_{ij}^2} = \sqrt{tr(\mathbf{S}^T\mathbf{S})}$. If $\mathbf{S}$ is a symmetric `symtensor` and n is the dimension, then $\rho(\mathbf{S}) \leq ||\mathbf{S}||_E \leq \sqrt{n}\rho(\mathbf{S})$ where $\rho(\mathbf{S})$ is the maximum magnitude eigenvalue of $\mathbf{S}$.

```
PLOT VARIABLES
   VELOCITY: MAGNITUDE
END
```

causes the magnitude of the velocity to be written, rather than each of the vector components. The EXODUS variable will have a "_MAG" appended at the end of the variable name, *e.g.*, VELOCITY_MAG.

For variables that vary by material, such as ENERGY, the default is to dump the values for each material separately as described above. The VOLUME AVERAGE conversion keyword causes the volume-weighted average to be dumped instead. Thus

```
PLOT VARIABLES
   STRESS: VOLUME AVERAGE
END
```

causes each stress component to be plotted as a single volume-averaged value, rather than as separate values for each material. The AVG keyword is an alias for VOLUME AVERAGE. Similarly, the MASS AVERAGE keyword produces a value for a multi-material element that is the mass weighted average of the quantity. Both VOLUME AVERAGE and MASS AVERAGE are applicable only to variables associated with a material, *e.g.*, PRESSURE or DENSITY. For both options the EXODUS file variable name will not have the "_N" pattern appended at the end of the name.

The MAXIMUM and MINIMUM keywords are entirely different conceptually. For material variables, a volume-weighted average is computed before proceeding. A more descriptive name would be MAXIMUM UP TO THE CURRENT TIME or MINIMUM UP TO THE CURRENT TIME. These maximum or minimum values may be positive or negative. In all other cases, such as vector and tensor variables, the MAGNITUDE (see above) is calculated, and then this scalar value is compared against the previous maximum or minimum value at that mesh location to obtain a new maximum or minimum over all simulation time. Thus maximum and minimum values will always be non-negative. These outputs will have "_MX" or "_MN" appended to the name. Note in particular for a symmetric tensor that the MAXIMUM and MINIMUM values are NOT min and max eigenvalues. For example, "STRESS_MX" is the maximum value of the Frobenius norm of the stress tensor up to the current simulation time at a given mesh point.

Thus

```
PLOT VARIABLES
   VOLUME
   VOLUME: MINIMUM
   DENSITY: AVG
   DENSITY: MINIMUM
   DENSITY: MAXIMUM
   VELOCITY
   VELOCITY: MAXIMUM
END
```

causes the volume and its minimum to be dumped, as well as the material-averaged density and its minimum and maximum, and the velocity and its maximum magnitude.

Users can override the default plot names using the syntax:

```
PLOT VARIABLES
   VELOCITY, AS, VEL
END
```

or if you find the second comma irritating,

```
PLOT VARIABLES
   VELOCITY, AS "VEL"
END
```

If the user is comparing old and new EXODUS files and wants precisely identical spellings of plot variables, the underscores that now separate the material and component tags can be removed in the variable names in the EXODUS plot file using the `NO UNDERSCORES` keyword. For example,

```
PLOT VARIABLES
   NO UNDERSCORES
   VELOCITY, AS "VEL"
END
```

which causes VELOCITY to be plotted as VELX, VELY, and VELZ, just as in some old EXODUS plot files formerly generated by ALEGRA.

Table 5: Plot Variables for General Region Quantities

Variable Name	Type	Explanation
COORDINATES	vector	nodal coordinates
MATCOOR	vector	original nodal coordinates
VOLUME	scalar	element volume
ASPECT RATIO	scalar	characteristic element length
MASS	scalar	nodal mass
EL MASS	scalar	element mass
PROC ID	scalar	for parallel runs, the processor id where the element's computations are done
VOID_FRC	scalar	fraction of an element's volume occupied by void
VOLFRC_n	material	fraction of an element's volume occupied by material n

Table 6: Plot Variables for Dynamic/Hydrodynamic Quantities

Variable Name	Type	Explanation
VELOCITY	vector	nodal velocity
FLUX	scalar	volume fluxes through element faces for REMAP
VAR VOLUME	scalar	variational volume, *i.e.*, element area, for 2D meshes.
VARIATIONAL FLUX	scalar	variational volume fluxes through element faces for REMAP on 2D meshes
REACTION*	vector	if REACTION enabled
REMESH*	vector	if REMESH enabled
Variables marked with an asterisk (*) cause ALEGRA to allocate additional dynamic memory for their storage.		

Table 7: Plot Variables for Hydrodynamic Quantities

Variable Name	Type	Explanation
MIDCOOR	vector	available only in unstructured mesh physics options
ACCELERATION	vector	nodal acceleration
DEFORMATION RATE	symtensor	symmetric part of velocity gradient
ARTIFICIAL VISCOSITY	tensor	The artificial viscosity term that is used to damp oscillations in the presence of shocks. The value is negative, *i.e.*, the forces will perform work on the material as opposed to the pressure which does work on the surrounding environment. It has dimensions of pressure.
HOURGLASS_RESISTANCE	vector	viscous term controlling hourglass modes
HOURGLASS_STIFFNESS	vector	stiffness term controlling hourglass modes
EXTERNAL ENERGY	scalar	energy from external sources
DETONATION TIME*	scalar	if PROGRAMMED BURN enabled
HE ENERGY ADDED*	scalar	if PROGRAMMED BURN enabled
HE TOTAL ENERGY*	scalar	if PROGRAMMED BURN enabled
Variables marked with an asterisk (*) cause ALEGRA to allocate additional dynamic memory for their storage.		

Table 8: Plot Variables for Material Properties

Variable Name	Type	Explanation
DENSITY	material	mass density
TEMPERATURE	material	material temperature
PRESSURE	material	material pressure
ENERGY	material	specific internal energy
ENERGY CHANGE	material	change in the specific internal energy during the last cycle
EQPS	material	equivalent plastic strain
SOUND SPEED	material	
SPECIFIC HEAT VOL	material	specific heat at constant volume
STRESS	symtensor	Cauchy stress tensor
STRETCH	symtensor	left stretch tensor
YIELD STRESS	material	yield stress in tension

4.2.13 History Plot Variables

```
HISTORY PLOT VARIABLES
   [NO MATERIAL GLOBALS]
   [NO UNDERSCORES]
   name [conversion]
   name [conversion]
   ...
END
```

This keyword permits the user to specify the node and element variables that will be associated with the tracers listed in the `TRACER POINTS` input (see Section 5.12 on page 137); data for these variables are ultimately written to the HISPLT plot database file [42]. By default (*i.e.*, the absence of a `HISTORY PLOT VARIABLES` section in the input file) all registered element and node variables for each tracer are written to the database, along with all global variables. If no `TRACER POINTS` are included in the input file, then only the global data will be written to the HISPLT plot file.

The guidelines for using the `HISTORY PLOT VARIABLES` input are the same as the `PLOT VARIABLES`, Section 4.2.12 on page 81, with the following exceptions.

In the output HISPLT files, the variable names will appear as requested, but will have minor differences because of the limitations of HISPLT. The "`outhis`" file produced by HISPLT, available after the first HISPLT test run, will list the exact variable names within the database that are available for display. These HISPLT variable names will be the same as the names included in the `HISTORY PLOT VARIABLES` keyword group, except that underscores within a name will be replaced by hyphens (or will be omitted if `NO UNDERSCORES` is designated in the keyword list), and the root name will be truncated to limit the string to 16 characters. The only global variable option available for history variables is to eliminate the plotting of the material global variables with the `NO MATERIAL GLOBALS` keyword.

For `vector` variables, such as `VELOCITY`, each vector component will be written to the HISPLT file with the name of the component direction appended. For example, the `x` component of velocity will be named `VELOCITY-X`.

For `material` tracer variables, the HISPLT name of the variable describ-

ing the quantity is indicated by appending the material id number to the `material` variable name. This is the number assigned by the user in the input file with the `MATERIAL` keyword. For example, the temperature of the material that is labeled 101 by the user is written to the HISPLT file with the name `TEMPERATURE-101`.

For `material` global variables, the HISPLT name of the variable describing the value of the quantity for a given material has nothing appended until the variable is requested in the HISPLT input, at which time the user appends the material number [42].

`tensor` quantities are labeled similarly, as for example `STRESS-XY`, for the `xy` component of the stress tensor in an element. `tensor` quantities that are specific to a `material` will have the material id appended to the variable name before the component designation. For example, `STRESS-202-XX` will be the name of the `xx` component of stress in material 202 in an element.

Valid values for the optional `conversion` keyword are:

```
{MAGNITUDE | VOLUME AVERAGE (AVG)}
```

If the `MAGNITUDE` of the `vector` or `tensor` quantity is requested, the component designators will be omitted. For names longer than 16 characters, including the material number and/or the component designation, it is recommended that the user override the default name to be consistent with the HISPLT database name length limitations.

If a material `VOLUME AVERAGE` (or `AVG`) quantity is requested, the material id will be omitted. In order for the HISPLT code to be able to distinguish the averaged variable name, *e.g.*, `TEMPERATURE`, from the name for the temperature for a specific material, *e.g.*, `TEMPERATURE-101`, *it is important for the user to specify the average quantity before the material specific quantity in the list of history plot variables*. For this temperature example, one should specify:

```
HISTORY PLOT VARIABLES
   ...
   TEMPERATURE: AVG   $ place the average request first
   TEMPERATURE        $ place the material specific request second
   ...
END
```

For all element variables, the value of the variable at the element center will be written to the database (*i.e.*, interpolation to the tracer location within the element is not performed for element variables). For nodal variables, interpolation to the tracer location is performed by default. If no interpolation is desired, this should be designated in the `TRACER POINTS` input section for each individual tracer (see Section 5.12 on page 137).

4.2.14 Restart Dumps

`RESTART DUMPS int (2)`

This keyword specifies the maximum number of restart dump files that an ALEGRA calculation can retain, `nmax`. These files will be named `runid.dmp.n`, where `runid` is the prefix of the ALEGRA input specification file, *i.e.*, the name used on the ALEGRA script execute line, and `n` is a file number. The value of `n` will start at 0 and increment by 1 each time a restart dump is written. When `nmax` files have been written, successive restart dumps will be matched by deletion of the earliest restart dump to keep the total number of restart dump files at `nmax`. By default, if no `RESTART DUMPS` keyword is used, `nmax` is 2 and only two restart dump files will be retained.

5 General Physics Input

A number of controls are available for setting up a problem, choosing the mesh, controlling the time step, setting various generic initial and boundary conditions, specifying how the mesh is to behave, and diagnosing material properties as the simulation proceeds. Section 5 describes some of the general features of ALEGRA that are applicable to nearly all types of physics.

Basic physics input includes:

- mesh choice (Section 5.1),
- physics specification (Sections 5.2 and 5.3),
- geometry specification (Section 5.5),
- time step control (Section 5.6),
- general initial conditions (Section 5.7),
- general boundary conditions (Section 5.8),
- block input (Section 5.9),
- domain input (Section 5.10),
- cell doctor input (Section 5.11),
- tracer points (Section 5.12), and
- function definition (Section 5.13).

5.1 Mesh Choices

It is important to understand some basic concepts about an ALEGRA mesh. An ALEGRA mesh is composed of a set of mesh `BLOCK`s which are all composed of the same type of elements. Currently only quadrilateral and hexahedral meshes have extensive support and testing in ALEGRA. This means that generally a mesh block simply refers to a separately numbered portion of the mesh. The mesh block is used to specify a focus region for initial conditions and Arbitrary Lagrangian-Eulerian (ALE) algorithm controls.

There are two types of mesh that can be used in ALEGRA. The choice of which mesh type is used is made by specifying a particular `physics` keyword. The two mesh types are unstructured and structured. The traditional mesh that ALEGRA has used is an unstructured, arbitrary connectivity mesh suitable for finite element methods. ALEGRA now also supports a structured mesh capability. A structured mesh is one in which the elements are arranged in a logically organized manner, similar to a two or three dimensional array. The mesh is still organized into blocks. Both rectilinear and curvilinear structured mesh is supported.

For the most part, the input for a structured mesh run is the same as for an unstructured mesh problem. Rather than have a separate manual section for a complete description of the structured mesh capability, and thus duplicate much of the information for unstructured, the differences from unstructured are described in Section 11 on page 192. In the rest of this manual, a section which contains a difference for structured mesh will be noted by a reference to Section 192.

5.2 Unstructured Mesh Physics Choices

Keywords specify the physics type(s) to use for a calculation. The options are listed in the following subsections.

5.2.1 Hydrodynamics

```
HYDRODYNAMICS
   ...
   [hydrodynamics keywords]
   ...
END
```

The choice of `HYDRODYNAMICS` as the physics specification will produce a simulation that results in deformation of the elements with zero stress deviators. Only the equations of state in the material model section can be used in these calculations.

5.2.2 Solid Dynamics

```
SOLID DYNAMICS
   ...
   [solid dynamics keywords]
   ...
END
```

The choice of `SOLID DYNAMICS` as the physics specification will produce a simulation that results in deformation of the elements with nonzero stress deviators. All of the available material models can be used in such a simulation.

5.3 Structured Mesh Physics Choices

These keywords specify the physics type(s) to use for a structured mesh calculation. The options are listed in the following subsections.

5.3.1 Structured Hydrodynamics

```
STRUCTURED HYDRODYNAMICS
   ...
   [structured hydrodynamics keywords]
   ...
END
```

The choice of `STRUCTURED HYDRODYNAMICS` as the physics specification will produce a simulation that results in deformation of the elements with zero stress deviators. Only the equations of state in the material model section can be used in these calculations.

5.3.2 Structured Solid Dynamics

```
STRUCTURED SOLID DYNAMICS
   ...
```

```
   [structured solid dynamics keywords]
   ...
END
```

The choice of STRUCTURED SOLID DYNAMICS as the physics specification will produce a simulation that results in deformation of the elements with nonzero stress deviators. All of the available material models can be used in such a simulation.

5.4 Multi-Region Dynamics

```
MRDYNAMICS
   SYNCHRONOUS
   ASYNCHRONOUS
   ACTIVATION SCHEDULE
      REGION int , [TIME real] | [CYCLE int]
      REGION int , [ [TIME real] | [CYCLE int] ] [SOURCE int]
      ...
      REGION int, [ [TIME real] | [CYCLE int] ] [SOURCE int]

   Region input sections
   ...

END
```

The choice of MRDYNAMICS as the physics specification allows multiple HYDRODYNAMICS and SOLID DYNAMICS physics to be analyzed in the same calculation, using both structured and unstructured meshes. The primary purpose of MRDYNAMICS is to provide a prototype for multiple region simulations. There is no coupling between regions or physics while the regions are running. Multiple separate problems may be run in separate regions. Each individual region uses the same keyword sequences as used for a standard single region problem. Each region will write plot output to its own output plot file, as determined by the PLOT keyword in the region input. For unstructured child regions, each region will use a genesis file specified by the MESH keyword in each region input section. There is no prohibition against mixing structured mesh and unstructured mesh regions within a multiregion dynamics problem.

5.4.1 Region Activation and synchronization

The `SYNCHRONOUS` keyword forces all regions to run at the same timestep. This will be the minimum timestep of all the regions currently active in the problem. Likewise, the `ASYNCHRONOUS` keyword allows all running regions to proceed in time independently with their own internally computed time steps. Using the `ACTIVATION SCHEDULE` keyword, it is possible to control the time or cycle at which a "child" region in the problem will start. The regions will terminate activity when the `TERMINATION TIME` or `TERMINATION CYCLE` keywords for each region has been exceeded.

5.4.2 Staged Activation of Regions

For unstructured mesh regions ONLY, it is also possible with the `SOURCE` keyword to specify the transfer of mesh data from one such region to another when the second region is activated. This process, called "staged region activation," allows the user to start a problem in a small spatial region, let it grow close to the boundary of that region and then transfer the results of the first region's calculations to a second region that models a larger spatial domain. The mesh of the second region must logically match the mesh of the first region. Also, the physics modeled in the two regions must be identical so that both regions have the same amount of data for an element and a node.

In problems that start in a very small region of space and then expand into the surrounding space, the user can take advantage of initial refinement in the first region to obtain a very highly resolved simulation. The first region can be initially refined to result in very small elements, thus improving the fidelity of the simulation in this very small space. Then, the `REDUCE MAX REFINEMENT` timed unrefinement capability (see Section 10.1.6 on page 189) can be used to reduce mesh refinement until the logical structure of the source mesh matches the level of refinement in the second "target" region. Data can then be transferred between the source and target meshes and the computation continued on the larger mesh.

This process can be continued indefinitely, allowing the calculation to expand to fill larger and larger regions of space with coarser and coarser mesh. However, there are certain practical limitations. In order to accomplish the transfer, both the target and source mesh and their data will reside in the

computer's memory at one time. Thus, for very large problems, there may not be sufficient memory available to perform the transfer.

5.5 Geometry

5.5.1 Cartesian

`CARTESIAN [int D]`

This keyword specifies that `CARTESIAN` geometry is to be used. It optionally specifies the dimensionality, which will be checked against the dimensionality of the executable used.

5.5.2 Cylindrical

`CYLINDRICAL [int D]` `(2D only)`

This keyword specifies that axisymmetric `CYLINDRICAL` geometry should be used. It optionally specifies the dimensionality, which will be checked against the dimensionality of the executable used. Cylindrical geometry assumes an `r-z` coordinate system with the radial direction plotted along the abscissa and the axial direction plotted along the ordinate. Note that cylindrical geometry is not supported by the 3D executable.

5.5.3 Volumetric Scale Factor

`VOLUMETRIC SCALE FACTOR real (1.) [SCALE LENGTH real (1.)]`

The `VOLUMETRIC SCALE FACTOR` keyword specifies a multiplier for global tallies such as mass and energy. This multiplier is a simple volumetric-based scaling factor. It may be useful for problems involving periodic boundary conditions to scale quantities by the degree of symmetry. It also may be useful for 2D Cartesian simulations to scale global quantities by the actual length of an object as opposed to assumed unit lengths of 1 m in `SI` units or 1 cm in `CGS` units (see the `UNITS` keyword in Section 4.1.3 on page 69). The

user is cautioned that this single factor may not correctly scale all surface tallies if there are multiple symmetries present (*e.g.*, translational, rotational, and mirror).

The `SCALE LENGTH` keyword is an optional additional multiplier. Internal to the code there is only one number. By use of this additional factor the user has more flexibility in exposing how the total multiplier is obtained.

This keyword was formerly known as the `SYMMETRY FACTOR` keyword.

For example,

```
volumetric scale factor 4. scale length .5
```

gives a total scale factor of 2. but the input syntax will emphasize that there is a four fold symmetry and an additional scale length factor of 0.5.

5.6 Time Step Control

5.6.1 Gradual Startup Factor

```
GRADUAL STARTUP FACTOR real (0.01)
```

This keyword specifies the factor by which the *initial* physics-based time step should be multiplied. This has the effect of gradually marching into an abrupt transient. This value should always be greater than zero and less than or equal to 1.0. See also the `MAXIMUM INITIAL TIME STEP` keyword.

5.6.2 Maximum Initial Time Step

```
MAXIMUM INITIAL TIME STEP real
```

This keyword permits the user to specify a maximum time step, otherwise ALEGRA computes an initial time step based upon the initial conditions. It is useful where unusual mechanical transients would otherwise result in an instability in the starting time step. For example, a small starting time step should be specified to permit the material a few cycles to begin responding

to energy deposition. Likewise, a large initial velocity on part of a mesh may require a small initial time step ($\Delta t_i <= maximum\_initial\_time\_step$). See also the `GRADUAL STARTUP FACTOR` keyword.

5.6.3 Maximum Time Step Limit

`MAXIMUM TIME STEP LIMIT real`

This keyword permits the user to specify a maximum time step value that the simulation will never exceed. Otherwise, the maximum time step is virtually unlimited ($\Delta t <= maximum\_time\_step\_limit$).

5.6.4 Maximum Time Step Ratio

`MAXIMUM TIME STEP RATIO real (1.2)`

This keyword sets the maximum ratio by which the time step may grow from one cycle to the next ($\Delta t_{n+1} <= \Delta t_n \times maximum\_time\_step\_ratio$).

5.6.5 Minimum Time Step

`MINIMUM TIME STEP real (1.0e-20)`

This keyword specifies the minimum time step permitted. If the stable time step is computed to be less than this value, the calculation will cease and write the final output records for a normal completion ($\Delta t >= minimum\_time\_step$).

5.6.6 Time Step Scale

`TIME STEP SCALE real (0.67 in 2D, 0.9 otherwise)`

The internal calculation of the maximum stable time step occasionally overestimates this quantity. The factors specified by this keyword allows

a smaller time step to be used. This option is particularly useful when instabilities appear; by setting these factors to small values, one may be able to distinguish physical from numerical instability.

This keyword causes the multiplication of the calculated time step by the specified real factor. The calculated time step is the minimum time step that has been selected from all other physics-based time-step constraints. Values less than 1 will shorten the time step, while values greater than 1 will increase the time step. Increasing the time step is not recommended since a loss of numerical accuracy may occur even though the numerical algorithms may be implicitly stable.

The following example input will cause the time step used in a calculation to be one half of the value ALEGRA would normally use.

```
time step scale 0.5
```

5.6.7 Constant Time Step

```
CONSTANT TIME STEP real
```

This keyword specifies that the simulation will be run with a constant time step value equal to the value specified here, otherwise ALEGRA determines a new time step each cycle. For example,

```
constant time step 1.0e-8
```

5.7 General Initial Conditions

5.7.1 Diatom

```
DIATOM
   PACKAGE name
      MATERIAL int
      INSERT shape
         insert subkeywords
      ENDI
```

```
      ...
      [additional package subkeywords]
      ...
   ENDP
   ...
   [additional diatom packages]
   ...
ENDDIATOM
```

The DIATOM capability is a method of inserting material into a mesh. The capability originates with the CTH [7, 25] Eulerian Solid Dynamics simulation code. *The DIATOM capability is intended for use only with rectilinear, orthogonal meshes aligned with coordinate axes.* The user is not prevented from using DIATOM with other meshes, but it must be recognized that sometimes very large inaccuracies will result from the approximation made for the cell volume. The DIATOM package assumes that the cell volume is a rectangle (or rectangular solid) defined by the minimum and maximum cell coordinates and aligned with the coordinate axes. For the CTH code, this would uniquely span the cell volume.

The input format for DIATOM in ALEGRA follows the DIATOM usage in CTH for the virtual objects package, which itself is modeled after the CTHGEN [1] input format for material insertion. The parsing of DIATOM keywords is done by the CTH DIATOM library, and therefore follows parsing rules for CTH. As a consequence, ALEGRA unit conversions (Section 4.1.3 on page 69) are NOT allowed in the DIATOM input section. Some functionality that exists in CTH is not currently available in ALEGRA. For example, the CTH virtual objects package allow objects to be inserted into the spatial mesh at times other than initialization. The functionality available in ALEGRA is listed in the tables below. Acceptable abbreviation limits are denoted by an asterisk (*) in the keyword.

There are two main types of keywords in the DIATOMs. The first set is entered after the initial DIATOM keyword, but outside of the PACKAGE ... ENDP keyword group. These properties will apply to all PACKAGEs in the set that precede the PACKAGE keyword. The DIATOM keyword set is ended by the ENDDIATOM keyword.

Table 9: Keywords for the DIATOM package.

Keyword	Argument	Meaning
XROT*ATE package(s) ENDX	real	Rotate all shapes about the x axis by this real value in degrees. The center of rotation is the mesh origin. The rotation affects all shapes listed between the keywords XROTATE and ENDX*ROTATE.
YROT*ATE package(s) ENDY	real	Rotate all shapes about the y axis by this real value in degrees. The center of rotation is the mesh origin. The rotation affects all shapes listed between the keywords YROTATE and ENDY*ROTATE.
ZROT*ATE package(s) ENDZ	real	Rotate all shapes about the z axis by this real value in degrees. The center of rotation is the mesh origin. The rotation affects all shapes listed between the keywords ZROTATE and ENDZ*ROTATE.
SCAL*E package(s) ENDS	real	Scale all shapes until the ENDS keyword by this real value.
TRA*NSLATE package(s) ENDT*RANSLATE	x, y, z	Translate shapes until the ENDT keyword by x, y, z.
PA*CKAGE ... ENDP*ACKAGE	string	Package name. This keyword begins a set of data that applies to all shapes specified within the PACKAGE/ENDP keyword group. The package name must be enclosed in single quotes if the following delimiters are within the package name: blank, comma (,), parentheses (), equal sign (=), or asterisk(*). A keyword identified in the table as "package subkeyword" must be within the PACKAGE input set to be recognized by the input parser. (Required keyword).

continued on next page

continued from previous page		
ENDDIA*TOM		The ENDDIA*TOM keyword is required to end material insertion input.

Most DIATOM inputs are grouped into PACKAGEs. All PACKAGEs are bracketed by a PACKAGE ... ENDP keyword group as indicated in Table 9. Multiple PACKAGEs are allowed and at least one PACKAGE is required.

There are a multitude of keywords that can be used to define the insertion of objects. Most of these keywords are optional. The minimum keywords required to define the insertion of an object are PACKAGE, MATERIAL, and INSERT. Note that only the blocks that specify ADD DIATOM INPUT will have the inserted object (see BLOCK main keyword). A complete set of the package keywords are provided in alphabetical order in Table 10 (ignoring any directional prefix).

PACKAGE subkeywords may be placed in one of two places. The keywords placed within a PACKAGE ... ENDP keyword group apply to all INSERTs until the corresponding ENDP keyword is reached.

The PACKAGE subkeywords determine:

- geometry (*e.g.*, INSERT or DELETE);
- the initial conditions of the material in the objects, (*e.g.*, MATERIAL id with DENSITY or XVELOCITY);
- the gradients of the initial conditions, (*e.g.*, AGRADED or CGRADED); and
- the initial resolution of the shape in the given mesh (*e.g.*, ITERATIONS or NUMSUB).

Examples are provided following the tables.

Table 10: PACKAGE Subkeywords for DIATOM Input

Subkeyword	Argument	Meaning
		continued on next page

continued from previous page

AGR*ADED P1=x,y[,z] P2=x,y[,z]	x, y, z x, y, z	Material properties in the package are graded axially away from P1 in the direction P2. The relevant material property must specify a table number; properties in the table should be defined as functions of the projection of $(x - x_1, y - y_1, z - z_1)$ from the vector $(x_2 - x_1, y_2 - y_1, z_2 - z_1)$. This feature can not be used in combination with CGRADED.
CGR*ADED P1=x,y[,z] P2=x,y[,z]	x, y, z x, y, z	Material properties in the package graded radially away from the axis defined by P1 and P2. The relevant material property must specify a table number; properties in the table should be defined as functions of the distance of $(x - x_1, y - y_1, z - z_1)$ from the vector $(x_2 - x_1, y_2 - y_1, z_2 - z_1)$. This feature can not be used in combination with AGRADED.
DEL*ETE shape ENDD*ELETE	string	Shape to be deleted (see allowable shapes and associated shape subkeywords in Table 11). After each shape, enter the required shape subkeywords. End each DELETE set with an ENDD keyword. The DELETE operations within a package are only effective on the shapes INSERT-ed within the same package.
DEN*SITY	real or T int	Initial density of the material in the insertion set and all subsequent insertions sets. The user must reset the density of subsequent sets to desired value or zero to obtain the proper initial ALEGRA values. A table number can also be input to specify density as a time varying function or for use with a graded option.

continued on next page

continued from previous page

INS*ERT shape ENDI*NSERT	string	Shape to be inserted (see allowable shapes and associated shape subkeywords in Table 11). After each shape, enter the required shape subkeywords. If the entire mesh is to be filled, a shape should extend beyond the mesh. End each INSERT set with an ENDI*NSERT keyword. Note that only the blocks that specify ADD DIATOM INPUT will have the inserted object (see BLOCK main keyword). The DELETE operations within a package are only effective on the material shapes INSERT-ed within the same package (required).
IT*ERATION	int	Number of iterations to recursively subdivide cell in each direction for insertion. Relates to CTH NUMSUB as follows: $ITER = nint(ln(NUMSUB)/ln(2.))$, where *nint* is the nearest integer function. Recommend ITER = 3, 4, or 5.
M*ATERIAL	int	Material Id of the next INSERT, DELETE, or REPLACE keyword to be inserted within this package. An individual package can have more than one material keyword if the user decides to operate on another material within the same package; however, most packages will logically contain only one material, which must be specified before the INSERT, DELETE, or REPLACE operation is specified (required).

continued on next page

continued from previous page

MV*ELOCITY	x, y, z	Material velocity vector for the object inserted. This velocity will apply to all subsequent insertions sets within the package unless replaced by another MV or RMV command.
N*UMSUB	int	In CTH, this is number of subdivisions per cell in each direction. In ALEGRA, the value given for NUMSUB will be used to calculate the ITERATION level as above.
RGR*ADED P1=x,y[,z]	x, y, z	Material properties in the package are graded radially about point P1. The relevant material property (density or temperature) must reference a table. Tabular properties should be defined as functions of $r = \sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2}$.
RMV*ELOCITY	x or T int	Radial material velocity for the object inserted. The direction of the velocity will be outward (positive x) or inward (negative x) from the point defined by the RGRADED command. Directionality is determined from the cell center and is transferred to the nodes; some inaccuracy is incurred in this so a relatively coarse mesh is not recommended.
STA*RT_ANYWHERE	none	This keyword is required if the problem start time is nonzero and an initial velocity is specified. Note that the underscore is required between the two words, or the keyword can be abbreviated to as few as three letters.

continued on next page

continued from previous page		
T*EMPERATURE	real or T int	Initial temperature of the material in the insertion set and all subsequent insertions sets. Must reset temperature of subsequent sets to desired value or zero to obtain the initial ALEGRA values. A table number can also be input to specify temperature as a time varying function or for use with a graded option.
XV*ELOCITY [YV*ELOCITY] [ZV*ELOCITY]	real	Initial Lagrangian velocity of the material in the insertion set and all subsequent insertions sets. The user must set velocity of subsequent sets to zero if no initial velocity desired. This velocity is superimposed on the material velocity vector (MVELOCITY or RMVELOCITY command).

A variety of shape options for the INSERT and DELETE keywords are provided. More shapes may be added as the need arises. The allowed shape options are listed in the Table 11. The shape options are a subset of those available in the CTH code.

Table 11: shape Subkeywords for DIATOM Input

Shape Keyword	Keyword or Numeric Input	Meaning
BOX	2D: P1 = x, y P2 = x, y 3D: P1 = x, y, z P2 = x, y, z	2D or 3D. The points P1 and P2 are the minimum and maximum coordinates of the box, aligned with the coordinate system.
CI*RCLE	CE*NTER = x, y R*ADIUS = r RI*NNER = r (0.)	2D only. The circle is defined by the CENTER and the RADIUS. An inner radius, RINNER, is optional.
		continued on next page

continued from previous page

UDS	P1 = x, y P2 = x, y P3 = x, y ... Pn = x, y	User Defined Shape – 2D only. This is an arbitrary shape defined by the points given. The first and last points will be connected by the diatom package to complete the shape. (Limited to 3000 coordinate points.)
CY*LINDER	CE1 = x, y, z CE2 = x, y, z R*ADIUS = r RI*NNER = r (0.)	3D only. The points CE1 and CE2 define the end points of the axis of the cylinder. The outer radius is given by RADIUS and the inner radius (optional) defined by RINNER.
OGIVE	CE1 = x, y, z CE2 = x, y, z OC*ENTER = uc, vc ORAD*IUS = r ORAN*GE = v1, v2	3D only. The points CE1 and CE2 define the end points of the axis of rotation. The OGIVE shape is defined in two-dimensional u-v coordinate space, where u = 0, v = 0 maps to CE1, and u = 0, v > 0 maps to CE2. OCENTER is the center of the OGIVE shape circle (uc < 0 for a proper OGIVE shape). ORADIUS is the radius of the OGIVE shape circle (r > -uc for proper OGIVE shape). ORANGE specifies the subset of the OGIVE shape to be inserted, given in points along the axis of rotation.
R2DP	CE1 = x, y, z CE2 = x, y, z P1 = u, v P2 = u, v ... Pn = u, v	Rotated 2D Polygon – 3D only. The points CE1 and CE2 define the end points of the axis of rotation. The points Pn define the shape along the axis, with v = distance along the axis, starting at CE1, and u = distance from the axis.

continued on next page

continued from previous page		
S*PHERE	CE*NTER = x, y, z R*ADIUS = r RI*NNER = r = 0	3D only. The sphere is defined by the CENTER and the RADIUS. An inner radius, RINNER, is optional.
TET*RAHEDRON	P1 = x, y, z P2 = x, y, z P3 = x, y, z P4 = x, y, z	3D only. The points define the vertices of the tetrahedron.
TO*ROUS	CE*NTER = x, y, z R*ADIUS = r P1 = x, y, z P2 = x, y, z	3D only. P1 and P2 define the axis of rotation for the circle of size RADIUS located at the CENTER given.
FNF	FILE = 'filename' MAT = n	Pro/Engineer tetrahedral mesh – 3D only. The insertions are to be read from an external file in the FNF format output from ProMesh. The filename must be enclosed in quotes, along with the material number to be extracted from the FNF file.

The following example illustrates the use of the DIATOM option.

```
block 1
 eulerian mesh
 add diatom input
end

diatom
   package cylinder_one
      material=34
      insert cylinder
         ce1 0. 10. 10.
         ce2 0. 10. 0.
         radius 5.0
         rinner 2.0
      endinsert
```

```
   endpackage


   translate  1. 1. 3.
   scale 0.5
   package ring
      material 34
      iteration 4
      yvelocity 1.e4
      insert torus
         center 4. 10. 10.
         radius 2.0
         p1 7. 0. 10.
         p2 7. 10. 10.
      endinsert
   endpackage

   package plate
      mat 2
      iteration 4
      insert r2dp
         ce1 0. 0. 10.
         ce2 13. 0. 10.
         p1 0. 0.
         p2 4. 3.
         p3 4. 10.
         p4 0. 13.
      endinsert
   endpackage
   ends
   endt

   package plate
      material 34
      insert fnf
         file='track.fnf' mat=2
      endi
   endp

$GRADED TEMPERATURE
```

```
   package 'part 1'
      temperature = t1
      rgraded p1 = 20., 10., 10.
      insert sphere
         ce = 20., 10., 10.
         r=11.
      endi
   endp
enddiatom

function 1
    0.   596.
   11.  1500.
end
```

5.7.2 User Defined Initial Conditions

```
USER DEFINED INITIAL CONDITION, variable
                              [, BLOCK int int ... ]
                              [, MATERIAL int int ... ]
"
  double v; double x = coord[0]; double y = coord[1];
  if ( fabs(x) < 1.e6 && fabs(y) < 1.e6 ) {
    v = 0.0;
  } else {
    v = sqrt( x * x + y * y );
  }
  field[0] = v;
"
END
```

This provides a general method for initializing known field variables on the mesh. The `variable` is a name, such as "density" or "velocity" which should not be quoted and is case insensitive.

As a C-language function, the quoted body has available one input array of coordinates, `coord`, and one output array of field values, `field`. The function is expected to use the coordinates in some way to set the return value. The coordinates is an array of length two for 2D and three for 3D, while

the return value is an array whose length depends on the type of variable. A scalar variable has length one, a vector has length 2 for 2D and 3 for 3D, etc.

The optional `BLOCK` keyword can be used to trigger the initial condition function only for certain element blocks, specified by their ids. This applies to nodal variables as well as element variables with one exception: there is no support for nodal variables on a block structured mesh on a per-block basis. Nodal variables that apply to all blocks *is* supported on both structured and unstructured meshes.

The optional `MATERIAL` keyword can be used to apply the initial condition function to only those elements which have the given material(s) present. Even nodal variables will not be set unless they touch an element with the material(s) present. The integers must specify a material id as specified in the input deck. The above mentioned exception applies here too: there is no support for nodal variables on a block structured mesh on a per-material basis. Nodal variables that apply to all elements regardless of material *is* supported on both structured and unstructured meshes.

Note that the C-language function has limited functions available to it and certain ANSI C syntax is not supported. Keep it simple and it should work. One special function that is provided is the `print()` function, which takes a variable name as its argument and prints to standard out the value of the variable.

5.8 General Boundary Conditions

5.8.1 Periodic Boundary Conditions

The ALEGRA user must supply an initial mesh which supports any periodic boundary conditions requested. This means that matching nodesets are required for each set of periodic boundaries. The nodes in the matching nodesets on the mesh boundary must be consistent with a periodic mesh. A necessary condition for the mesh to be allowable is that the number of nodes in the matching nodesets must be the same. Each node in one nodeset corresponds one-to-one to a node in the matching nodeset. The user may pick either a translation or a rotation for each periodic boundary condition. Details of the current implementation of periodic boundary conditions in

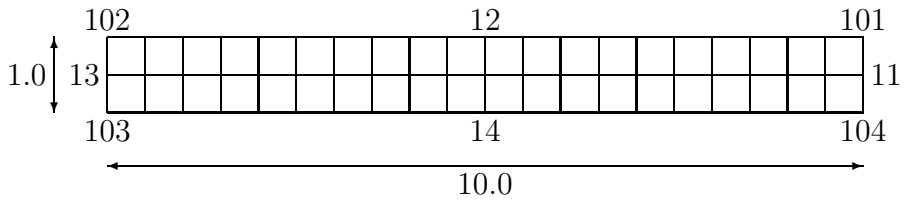

Figure 8: Periodic mesh example.

ALEGRA can be found in Reference [28]. Structured mesh periodic boundary conditions are described in Section 11 on page 192.

5.8.2 Translational periodicity

```
PERIODIC BC, nodeset1, TRANSLATE vector, nodeset2
             [TOLERANCE real (1.e-5)]
```

where `nodeset1` and `nodeset2` are the first and second nodesets, `vector` is the translation vector $\vec{u}$ which maps the nodes in `nodeset1` to `nodeset2`. That is,

$$\vec{x}_2 = \vec{x}_1 + \vec{u} \tag{5.1}$$

Example:

```
periodic bc, nodeset 3, translate, x 1. y 0., nodeset 1
```

A 2D Cartesian mesh may be made doubly periodic by associating the two pairs of opposite nodesets and the two pairs of diagonally opposite corners. For example, suppose the opposite sides are labeled by nodesets 11 to 14, and the four corner vertices are labeled by nodesets 101 to 104 (each of these latter nodesets containing only one node), and the mesh is 10 units by 1 unit, as shown in Figure 8.

The input to make this mesh doubly periodic in X and Y are:

```
periodic bc, nodeset  13, translate, x 10. y  0., nodeset  11
periodic bc, nodeset  14, translate, x  0. y  1., nodeset  12
```

```
periodic bc, nodeset 103, translate, x 10. y  1., nodeset 101
periodic bc, nodeset 102, translate, x 10. y -1., nodeset 104
```

A similar method may be used to make a 3D Cartesian mesh triply periodic. In this case one must associate the three pairs of opposite faces, the six pairs of diagonally opposite (and parallel) edges, and the four pairs of diagonally opposite corners. All associations pass through the center of the rectangular parallelepiped.

5.8.3 Rotational Periodicity

```
PERIODIC BC, nodeset1, ROTATE real, POINT vector1,
             AXIS vector2, nodeset2
             [TOLERANCE real (1.e-5)]                  (3D)

PERIODIC BC, nodeset1, ROTATE real, POINT vector1,
             nodeset2
             [TOLERANCE real (1.e-5)]                  (2D)
```

`vector2` is the rotation axis passing through the point `vector1` which rotates the nodes in `nodeset1` through an angle given in degrees to coincide with the nodes in `nodeset2`. In 2D, this boundary condition only makes sense for Cartesian geometry where the axis of rotation is orthogonal to the mesh.

3D example:

```
periodic bc, nodeset 3,
             rotate  45.,
             point,  x 0. y 0. z 0.,
             axis,   x 0. y 0. z 1.,
             nodeset 4
```

2D example:

```
periodic bc, nodeset 3, rotate 45., point, x 0. y 0., nodeset 4
```

5.9 Block Input

```
BLOCK int
   [subkeyword-list]
END
```

The BLOCK keyword group allows the user to specify the materials that are contained in a block and the type of mesh movement desired in the block. The default is for a block to be a voided Lagrangian block (*i.e.*, if all subkeywords are omitted). The BLOCK subkeywords are described in the tables below.

It is important to understand some basic concepts about an ALEGRA mesh. An ALEGRA mesh is composed of a set of mesh BLOCKs which are all composed of the same type of elements. Currently only quadrilateral and hexahedral meshes have extensive support and testing in ALEGRA. This means that generally a mesh block simply refers to a separately numbered portion of the mesh. The mesh block is used to specify a focus region for initial conditions and Arbitrary Lagrangian-Eulerian (ALE) algorithm controls.

SMALE (Single Material ALE) refers to ALE remeshing within a single block in the sense that no MATERIAL is allowed to flow across the Lagrangian block boundaries. In ALEGRA, SMALE does not imply that only a single material is present within the block since ALEGRA can be always be initialized with multiple materials in each element. The single material part of the SMALE name has its roots in the early technique or concept of block material initialization in which a block was initialized to a single material. A better name today would be Single Block ALE, but the SMALE name is till in use.

An MMALE (Multi-Material ALE) designation allows two adjoining mesh blocks to be treated as a single large ALE block. Thus MMALE blocks do not necessarily have Lagrangian boundaries. The name Multi-Block ALE is more precise, but the MMALE name is currently used. Two adjoining MMALE blocks are treated as a single ALE region and MATERIAL is allowed to flow between these two blocks.

Additional ALE designators can be given on sidesets and nodesets to precisely define the ALE algorithms intended on BLOCK boundaries.

A special BLOCK keyword is available for structured mesh physics options

as described in Section 11 on page 192.

5.9.1 Material Specification

A principal use of the BLOCK is to insert materials into the mesh. This is performed on a block by block basis. ALEGRA will process DIATOM first and the MATERIAL input last.

If multiple DIATOM packages affect the same block, they will be processed in the order they appear in the input deck. If two or more DIATOM shapes associated with the packages overlap in the same mesh cell, ALEGRA will compute a volume fraction for the material associated with the first PACKAGE and the remainder of the cell volume will be temporarily assigned to void. The volume fraction is determined from the intersection of the shape with the volume of the cell. Next ALEGRA will compute a volume fraction for the material associated with the second PACKAGE and use the minimum of this volume fraction and the void fraction as the available volume fraction for the second material. The volume fraction for the second PACKAGE is computed as if there is no knowledge of the first PACKAGE. It is only the available void fraction that provides one PACKAGE with a knowledge of a previous PACKAGE. The process continues until all DIATOM are exhausted. Overlaps can be used to the user's benefit to prevent minuscule amounts of void or stray material from creeping into cells due to round off error in computing volume fractions on irregular meshes. The user is cautioned, however, that the results may be order dependent and the results of multiple DIATOM inserts should be carefully checked.

If both the ADD DIATOM INPUT and the MATERIAL keywords are present in the same block, all of the DIATOM will be processed first. Any remaining void fraction will be replaced with the material specified on the MATERIAL keyword.

If multiple MATERIAL keywords are present in given block, then the available volume fraction is equally divided among the materials specified.

Example:

```
block 1
   add diatom input  $ diatoms are processed first
   material 1        $ mat 1 fills remainder of block
```

Table 12: BLOCK MATERIAL Initialization Keywords.

Sub-Keyword	Input	Meaning
MATERIAL	int	Value corresponds to an id specified by a valid MATERIAL keyword. There can be multiple material entries.
ADD DIATOM INPUT		Indicates that volume fractions will be calculated in this BLOCK from the information given in the DIATOM keyword group. The materials are identified in the DIATOM keyword group input.

```
   lagrangian mesh
end
```

5.9.2 Mesh Type

A second principal use of the BLOCK input is specification of the type of mesh. Valid mesh types are listed in Table 13. The specification of LAGRANGIAN MESH, SMALE MESH, MMALE MESH, or EULERIAN MESH initially sets the mesh movement for the block to these types. A question arises regarding how to treat the mesh nodes on the boundary between blocks of two different mesh types. By default, ALEGRA assumes a certain hierarchy of mesh types.

```
EULERIAN < MMALE < SMALE < LAGRANGIAN
```

ALEGRA assigns the greater type from this hierarchy to the nodes on a common boundary. However, particular nodes within the mesh may be changed to a different type depending on nodeset/sideset definitions and on the blocks that neighbor this particular block. See the DOMAIN keyword for more information on how to use the nodeset/sideset keywords for specific REMESH control.

Example:

```
block 1
   material 10
```

Table 13: BLOCK Mesh Specification Keywords.

Sub-Keyword	Meaning
LAGRANGIAN MESH	In the Lagrangian formulation the nodes move at the material velocity. This is the default mesh type.
SMALE MESH	In a single-material (single block) ALE formulation, the mesh moves with the material until the distortion becomes large enough to trigger the remeshing of the nodes and a corresponding remapping of material quantities. Material cannot flow out of the mesh BLOCK.
MMALE MESH	In a multi-material (multi-block) ALE formulation, the mesh moves with the material until the distortion becomes large enough to trigger the remeshing of the nodes and a corresponding remapping of material quantities. Material may flow out of the mesh BLOCK.
EULERIAN MESH	In an Eulerian formulation, the nodes remain fixed.
SMOOTHED EULERIAN MESH	In this formulation, the nodes are repositioned to their original coordinates (as for EULERIAN MESH), but the mesh is then smoothed.

```
   smale mesh
end

block 2
   material 20
   lagrangian mesh
end
```

5.9.3 Remap Control

Another important function of the BLOCK input is to control the remap or advection. BLOCK keywords are used in conjunction with various DOMAIN

keywords to produce the desired level of control over mesh behavior. An ALEGRA simulation cycle consists of a Lagrangian step where the material and the mesh move together. If the mesh type is not Lagrangian, then the mesh nodes are moved back to their former positions if the mesh is Eulerian, or perhaps to some other position if the mesh is `ALE`. This movement of nodes is termed the remesh phase of a remap. When the mesh nodes are moved, the material position must remain fixed. Thus, a certain fraction of the material in a remeshed cell must be transferred to adjacent cells depending on the amount of volume that fluxes or passes through each element face. This fluxing of material is known as advection. In ALEGRA, "remapping" means mesh movement or "remeshing" followed by advection.

Several remesh methods are available and are listed in Table 14. These remesh methods move nodes interior to the mesh. The boundary nodes remain fixed. Therefore, there must be interior mesh nodes for remeshing to occur. A consequence of this is that in simple 1D-like problems, where there is only one row of mesh cells, nodes cannot be remapped. (1D-like problems may be used for simple scoping studies.) The frequency of remaps is also controlled by the `REMESH FREQUENCY` keyword in Table 14.

The `AVERAGE REMESH METHOD` moves a node to the average of the element centers to which the node is connected, although it produces instabilities in some test cases. The `BUDGE REMESH METHOD` attempts to produce an orthogonal mesh.

The `TIPTON REMESH METHOD` solves the inverted Laplace equation through a variational approach. The resulting set of equations are solved using Jacobi iteration. It is unnecessary to find the exact solution to these equations. Excessive node movement may over empty a cell. (See `REMESH MOVEMENT` controls.) This algorithm without weights attempts to equalize the volumes of the elements about a node. With weights, one can move nodes toward regions of interest. The `WINSLOW REMESH METHOD` is the `TIPTON REMESH METHOD` without weights.

Remeshing is a multi-step process. The first step is to determine which nodes in the mesh are eligible to be remeshed. Lagrangian nodes are not remeshed. Eulerian nodes are remeshed to their former positions, prior to the Lagrangian step. `ALE` nodes are conditionally remeshed depending on whether some threshold condition is met. If the condition is not met then the node is not remeshed and the mesh behaves in a Lagrangian manner. If the condition is met, then the node is remeshed and the mesh behaves in

Table 14: BLOCK Remesh Methods Sub-Keywords.

Sub-Keyword	Input	Meaning
AVERAGE REMESH METHOD BUDGE REMESH METHOD TIPTON REMESH METHOD WINSLOW REMESH METHOD		Default = TIPTON. Turns on the indicated interior remesh method.
REMESH FREQUENCY	int (1)	Number of time steps between a remap step.
EULERIAN MOVEMENT {X | Y | Z}		Specifying X, Y, or Z cause the mesh to remain fixed relative to material motion in that direction.
RADIAL CONSTRAINT		Constrains remesh movements to the radial direction. (Assumed center at (0,0,0) for now.

an ALE manner. SMOOTHED EULERIAN nodes are treated as Eulerian unless a remesh condition is met. When the remesh conditions are satisfied they are remeshed just as an ALE node would be. While ALE nodes will behave in a Lagrangian manner unless the remesh conditions are triggered, the SMOOTHED EULERIAN nodes will behave in an Eulerian manner unless the conditions for remesh are met. The threshold conditions are known as triggers.

Triggers may be based on geometric or physical considerations as described in the Table 15.

Table 15: BLOCK Remesh Trigger Sub-Keywords.

Sub-Keyword	Input	Meaning
ANGLE TRIGGER	real	Minimum node angle on which to trigger a nodal remesh. This value is entered as the absolute value of the cosine of the minimum angle. An ideal angle is 90^o or a value of zero for the cosine of the angle. A value of zero will guarantee that every node is remeshed. Note that this trigger is NO LONGER ON BY DEFAULT.
continued on next page		

continued from previous page

SOLID ANGLE TRIGGER	real	Trigger a nodal remesh based on the minimum solid angle at a node. This value is entered as a ratio, which should not be exceeded, of an ideal solid angle to the minimum solid angle. The ideal solid angle is the total interior solid angle (4π for an interior node) divided by the number of elements connected to the node. A value of 1.0 will guarantee that every node is remeshed.
VOLUME TRIGGER	real	Minimum adjacent element volume on which to trigger nodal remesh. A value of 1.0 will guarantee that every node is remeshed. Note that this trigger is NO LONGER ON BY DEFAULT.
VARVOL TRIGGER	real	Analogous to the volume trigger, but uses the "variational volume" which in 2D cylindrical geometry is the area of the element.
DENSITY TRIGGER	real	This trigger will flag nodes for remeshing when the average density in its associated elements is *greater than or equal to* the input threshold value. The average is a simple unweighted numerical average of the density values.
TEMPERATURE TRIGGER	real	This trigger will flag nodes for remeshing when the average temperature in its associated elements is *greater than or equal to* the input threshold value. The average is a simple unweighted numerical average of the temperature values.
PRESSURE TRIGGER	real	This trigger will flag nodes for remeshing when the average pressure in its associated elements is *greater than or equal to* the input threshold value. The average is a simple unweighted numerical average of the pressure values.
INVERSE DENSITY TRIGGER	real	This trigger will flag nodes for remeshing when the average density in its associated elements is *less than or equal to* the input threshold value. The average is a simple unweighted numerical average of the density values.

continued on next page

continued from previous page		
INVERSE TEMPERATURE TRIGGER	real	This trigger will flag nodes for remeshing when the average temperature in its associated elements is *less than or equal to* the input threshold value. The average is a simple unweighted numerical average of the temperature values.
INVERSE PRESSURE TRIGGER	real	This trigger will flag nodes for remeshing when the average pressure in its associated elements is *less than or equal to* the input threshold value. The average is a simple unweighted numerical average of the pressure values.
COMBINE TRIGGERS		This option causes triggers to be applied in conjunction, *i.e.*, logically and-ed together. The normal default is for the triggers to operate separately, *i.e.*, logically or-ed together.

As described above, the TIPTON method solves the inverted Laplace equation through a variational approach. Using this method with weights, one can move nodes toward regions of interest. Weight can be based on geometric or physical considerations. Various weights are described in the following table. The weight used in the TIPTON method is the maximum of the computed value or the threshold value. Node movement is biased toward regions of the mesh where the computed weight exceeds the threshold. Regions of the mesh where the weight value is below the threshold are equally weighted. *The TIPTON scheme must be used in all blocks that share a remeshed node.* If this is not done, the remeshed node on the block boundary will migrate rapidly toward the interior of the block.

Once the weights associated with a given method are computed, these weights are then scaled to the range 1.0 to NORMALIZATION FACTOR. This prevents excessive node movement. Each weighting method is individually normalized. The normalization may be linear or logarithmic and is specified after the weight keyword. For example, in some explosive or ablative simulations the density may vary over several orders of magnitude. In this case it may be wise to use a logarithmic normalization to capture the wide range of densities.

Normalizing each weight also puts each weight on an equal basis should multiple weights be specified in a single BLOCK. Multiple weights are linearly summed after each is individually scaled. The resultant sum is then linearly

rescaled to the range 1.0 to NORMALIZATION FACTOR. The relative importance of multiple weights can be controlled through judicious choice of the various thresholds.

Table 16: BLOCK Remesh Weight Sub-Keywords.

Sub-Keyword	Input	Meaning
DENSITY WEIGHT DENSITY GRADIENT WEIGHT TEMPERATURE WEIGHT TEMPERATURE GRADIENT WEIGHT PRESSURE WEIGHT PRESSURE GRADIENT WEIGHT INVERSE VOLUME WEIGHT INVERSE RADIUS WEIGHT INVERSE XYRADIUS WEIGHT INVERSE YZRADIUS WEIGHT INVERSE XZRADIUS WEIGHT	real	Any combination of these keywords can be entered. Their entry turns the specific type of weighting on for BLOCKs specified with Tipton smoothing. The numeric input (real $\geq$ 0.) serves as a threshold to control application of the weight. The weight is applied when the element quantity exceeds the threshold.
INVERSE DENSITY WEIGHT INVERSE TEMPERATURE WEIGHT INVERSE PRESSURE WEIGHT	real	Any combination of these keywords can be entered. Their entry turns the specific type of weighting on for elements specified with Tipton smoothing. The numeric input (real $\geq$ 0.) serves as a threshold to control application of the weight. The weight is applied when the inverse of the element quantity exceeds the inverse of the threshold.

continued on next page

continued from previous page		
`choice_of_weight real {LINEAR NORMALIZATION | LOG NORMALIZATION}`		Default = `LINEAR`. These two keywords modify the above weights and determine the method used to normalize weights for Tipton smoothing. Weights are normalized between 1.0 and the value given by the `NORMALIZATION FACTOR` keyword.
`NORMALIZATION FACTOR`	`real` (4.0)	Maximum weight value (`real` > 1). Larger values may cause extreme mesh movement and over-fluxing of an element. If `real` = 1.0, weighting is ineffective.

Example:

```
block 1
   material 7
   smale mesh
   tipton remesh method

   density trigger = 30.0
   density weight = 50.0 $ defaults to linear scaling

   temperature trigger = 3000.
   temperature weight = 5000. linear norm

   inverse density trigger = 0.01
   inverse density weight = 0.005 log norm

   norm factor 7.0 $ allow greater node movement
end
```

This example would trigger remeshing in regions of the mesh where there are high densities and temperatures or low densities, leaving regions of the

mesh with intermediate values alone. The inverse density weight uses a logarithmic normalization.

The user can also control the method used to advect material variables between elements and momenta between nodes. The keyword is given in Table 17. *Note that these keywords are operative for structured mesh physics options since the Eulerian capability is supported.*

Table 17: Advection Control Sub-Keywords.

Sub-Keyword	Input	Meaning

`DONOR ADVECTION`
`SUPERB ADVECTION`
`VANLEER ADVECTION`
		Default = `VANLEER`. Turns on the indicated advection method.

5.9.4 Other Block Controls

Other parameters and behaviors can be controlled within the `BLOCK` keyword, including artificial viscosity, hourglass control, and mesh modifications. See Table 18.

Table 18: Other `BLOCK` Sub-Keywords.

Sub-Keyword	Input	Meaning
`ARTIFICIAL VISCOSITY`	`int`	Specifies the artificial viscosity model to be used for this block (see `ARTIFICIAL VISCOSITY` keyword input).
`PRESCRIBED {X | Y | Z} VELOCITY`	`real`	Apply a fixed non-zero velocity to the nodes in this block. Valid for `EULERIAN MESH` and `SMOOTHED EULERIAN MESH` blocks.
`HOURGLASS CONTROL`	`int`	Specifies the hourglass control model to be used for this block (see `HOURGLASS CONTROL` keyword input).
continued on next page		

continued from previous page		
DELETION CYCLE	int	Specifies the cycle at which the element block is deleted from the problem.
DELETION TIME	real	Specifies the time at which the element block is deleted from the problem. Time must be greater than or equal to DELETION TIME to trigger deletion.
DELETE DATA		Delete element block by deleting all vertex, edge, face, and element data associated with the block. Note that coordinates are reset to original value and entire block is filled with void.
DELETE TOPOLOGY		Delete element block by deactivating all vertices, edges, faces and elements associated with the block.

5.10 Domain Input

```
DOMAIN
   [subkeyword-list]
END
```

The DOMAIN keyword group allows the user to specify how an entire domain is to behave, *i.e.*, global behavior that cannot be broken down to the block level. The following tables describe the allowed subkeywords that can be specified for a domain.

5.10.1 Boundary Remesh Control

In order to properly use the sideset and nodeset specifications available in the DOMAIN input, a few key points must be understood about the functional characteristics of the ALEGRA rezone method. By using the word "rezone", this discussion is limited to the methods ALEGRA uses to reposition nodes for the ALE method.

The first basic understanding needed is that, unless told differently by the user, ALEGRA will only rezone *interior* nodes in a mesh block. The user *must* tell the code that rezoning is desired on the sides of these blocks, using the sideset and nodeset keyword commands available in `DOMAIN`. The sideset rezone methods will compute new locations for nodes in the sideset that are interior to the side, *i.e.*, not located on the edges outlining the side. To control rezone of the nodes on the edges, the user *must* employ the nodeset keyword commands available in `DOMAIN`.

The next point that must be understood is the method ALEGRA uses to resolve overlapping specifications for nodes on sides and edges. A node is considered to belong to up to four entities: a block, a sideset, a nodeset and a pointset (a nodeset with only a single node included). The node can also have one of four rezone characteristics: `LAGRANGIAN`, `ALE`, `SMOOTHED EULERIAN` or `EULERIAN`. These characteristics are set by the keyword commands available in the `BLOCK` and `DOMAIN` inputs.

The `BLOCK` mesh specification (*e.g.*, `EULERIAN MESH`) will label the nodes in the mesh block in a manner that depends upon the type of block and the whether the node is in the interior of the block or not. Table 19 explains this labeling.

Table 19: Node Rezone Control by Mesh Specification

Block Type	Interior Nodes	Non-Interior Nodes
`LAGRANGIAN`	`LAGRANGIAN`	`LAGRANGIAN`
`SMALE`	`ALE`	`LAGRANGIAN`
`MMALE`	`ALE`	`ALE`
`SMOOTHED EULERIAN`	`SMOOTHED EULERIAN`	`SMOOTHED EULERIAN`
`EULERIAN`	`EULERIAN`	`EULERIAN`

When conflicts between block labels occur, as when nodes are common to two or more blocks, a precedence of node labels is enforced. The node attributes are enforced in the following manner:

```
LAGRANGIAN > ALE > SMOOTHED EULERIAN > EULERIAN.
```

The ">" symbol means "overrides". Thus a node will have the most restrictive setting applied to it by multiple overlapping specifications.

The next specification, allowed in 3D problems only, is a sideset specification. For a sideset, the distinction between types of nodes are those that are on a material interface and those that are not on such an interface. In this context the nodes on each sideset are labeled according to the rules explained in Table 20.

Table 20: Node Rezone Control by Sideset Specification

Sideset Type	Non-Interface Nodes	Interface Nodes
LAGRANGIAN	LAGRANGIAN	LAGRANGIAN
SMALE	ALE	LAGRANGIAN
MMALE	ALE	ALE
SMOOTHED EULERIAN	SMOOTHED EULERIAN	SMOOTHED EULERIAN
EULERIAN	EULERIAN	EULERIAN

Once again the node label precedence is enforced for nodes that share one or more sidesets.

The nodesets are processed and labeled in a manner that is exactly the same as the sidesets, with the distinction of node type being the presence of the node on a line where a material interface intersects a mesh boundary. Finally, the point sets are processed, these being the one-entry nodesets.

Once all of the above has been done, the code has a block label, a sideset label (if 3D), a nodeset label and a point label for each node in each block. Note that the latter three labels will only be present if the user specified the node by means of node or sidesets. Each node will however have a label from the block specification. The final step in labeling the nodes to allow remesh motion is to allow the following node movement enforcement.

```
Point type > Nodeset type > Sideset type (if 3D) > Block type.
```

Here, the ">" symbol indicates "overrides". With these controls, the user can determine how the ALEGRA rezone package will operate within the mesh.

Table 21: `DOMAIN` Keywords for Remesh Control

Sub-Keyword	Input	Meaning
		continued on next page

continued from previous page

LAGRANGIAN NODESET EULERIAN NODESET SMALE NODESET SMALE XLINE NODESET SMALE YLINE NODESET SMALE ZLINE NODESET MMALE NODESET MMALE XLINE NODESET MMALE YLINE NODESET MMALE ZLINE NODESET SMOOTHED EULERIAN NODESET SMOOTHED EULERIAN XLINE NODESET SMOOTHED EULERIAN YLINE NODESET SMOOTHED EULERIAN ZLINE NODESET	int	Value corresponds to a nodeset id and indicates how the nodes of the set should behave relative to the material motion. The DOMAIN keyword can have multiple entries of this type. Note: The SMALE, MMALE, and and SMOOTHED EULERIAN features produce slightly different results in parallel runs than serial runs. The LAGRANGIAN, EULERIAN, SMALE, MMALE, and SMOOTHED EULERIAN NODESET commands control the setting of nodal flags on the nodes of the nodeset.
LAGRANGIAN SIDESET EULERIAN SIDESET SMALE SIDESET SMALE XYFACE SIDESET SMALE YZFACE SIDESET SMALE XZFACE SIDESET MMALE SIDESET MMALE XYFACE SIDESET MMALE YZFACE SIDESET MMALE XZFACE SIDESET SMOOTHED EULERIAN SIDESET SMOOTHED EULERIAN XYFACE SIDESET SMOOTHED EULERIAN YZFACE SIDESET SMOOTHED EULERIAN XZFACE SIDESET	int	Value corresponds to a sideset id and indicates how the nodes of the set should behave relative to the material motion. The DOMAIN keyword can have multiple entries of this type. The LAGRANGIAN, EULERIAN, SMALE, MMALE, and SMOOTHED EULERIAN SIDESET commands control the setting of nodal flags on the nodes of the sideset. Note: The XYFACE, YZFACE, XZFACE commands must be used in order to remesh the nodes in the sideset.
REMAP ITERATIONS	int (1)	Number of remaps to perform at the end of a Lagrangian step.
REMESH ITERATIONS	int (10)	Number of remesh smoothing iterations per remap.

continued on next page

continued from previous page		
INITIAL REMESH MOVEMENT LIMITER	real(1.0)	Fraction of calculated smoothing movement to allow at problem startup. $0 \leq$ real ≤ 1. Use this to avoid large mesh movements at problem startup which can cause overfluxing in the advection phase of the remap.
REMESH MOVEMENT RATIO	real(1.0)	Change remesh movement limiter by this factor each time through a remesh smoothing iteration. real ≥ 1.
REMESH MOVEMENT LIMITER	real(1.0)	Maximum fraction of calculated remesh movement to allow. $0 \leq$ real ≤ 1.

5.10.2 Domain Advection Controls

The DOMAIN keyword group also holds controls for the advection methods used by ALEGRA. These are given in Table 22.

Table 22: DOMAIN Keywords for Advection Control

Sub-Keyword	Input	Meaning
SALE ADVECTION MSALE ADVECTION SHALE ADVECTION MSHALE ADVECTION HIS ADVECTION MHIS ADVECTION		Default = HIS ADVECTION. Turns on the indicated nodal advection method. These are documented in detail elsewhere [26].
SLIC INTERFACE TRACKER SMYRA INTERFACE TRACKER NEW SMYRA INTERFACE TRACKER		Default = SMYRA INTERFACE TRACKER. Turns on the indicated material interface tracker. SLIC is present only for testing purposes and is not supported for structured mesh physics options. NEW SMYRA is a recent modification of the algorithm by R. Bell.
		continued on next page

continued from previous page		
TOTAL ENERGY ADVECTION		Total energy is advected and conserved during advection. The default capability is to advect and conserve internal energy during advection.
VOIDED SIDESET	int	Value corresponds to a valid sideset id and indicates that void exists on the other side of this Eulerian mesh boundary. By default, all Eulerian, Ale and Smoothed Eulerian mesh boundaries are assumed to reflect their conditions on the exterior of a problem. By default, all Lagrangian boundaries are free surfaces.

The VOIDED SIDESET command controls the inflow/outflow of material from the mesh. By default, the material state outside the mesh will be the same as the material state just inside the mesh boundary. By using VOIDED SIDESET, the user can tell ALEGRA that there is void outside the mesh and material can leave the mesh or void can enter the mesh. In using this capability, the Lagrangian boundary conditions control the motion of the material. So, if there is a NO DISPLACEMENT boundary condition on the mesh boundary that inhibits motion normal to the boundary, then the VOIDED SIDESET command will have no effect on advection.

The nodal advection method concerns the advection of momentum between nodes of the mesh. All other advection concerns element centered quantities, and advection is a matter of finding a value for the quantity on an element face and then moving a volume of the quantity between elements in a conservative manner. For momentum however, the ALEGRA velocities are centered on nodes and there is no convenient "face" to advect momentum through. Thus some method must be used to find the momentum associated with a node, advect this value and then redistribute to the nodes.

The interface tracker determines the location of the material interfaces within an element so advection can move the appropriate material into or out of a face. There is actually only one choice here. The SLIC INTERFACE TRACKER leads to spurious "streaming" effects in regular mesh. It is present in the code only for testing purposes.

Advecting total energy is an option available to the user. The need for

this option arises because it is not possible to simultaneously conserve mass, momentum, internal energy and kinetic energy during advection. ALEGRA is designed to always conserve mass and momentum. The momentum conservation determines the post-advection nodal velocities and these will produce a kinetic energy that when added to the conserved post-advection internal energy, will result in a total energy that will not exactly equal the pre-advection value. This effect is aggravated by advection errors caused by steep gradients and/or coarse zoning.

Since some problems require exact total energy conservation, an option is available that results in conservation of total energy rather than internal energy. By selecting this feature, the specific kinetic energy is added to the specific internal energy of each material in an element prior to advection. This quantity is conservatively advected, and after remap the specific kinetic energy is subtracted from the result. The specific kinetic energy is the average over the nodes of an element of the quantity $\frac{1}{2}v^2$.

Use of the total energy advection option can have bad consequences. Negative internal energies or spuriously large internal energies (artificial heating) can result from this option. These effects will occur when the momentum advection across an element side implies a kinetic energy flux that does not match the internal energy flux across the element side. The final accounting will place any deficit or surplus in the element's internal energy.

5.10.3 Initial Refinement

```
INITIAL REFINEMENT
   [subkeyword-list]
END
```

The `DOMAIN` keyword group also contains controls for the initial refinement capability of ALEGRA. Using this feature, a user can specify that a quadrilateral or hexahedral initial mesh can be refined prior to the problem starting. Each level of refinement multiplies the number of elements by a factor of 4 in 2D and 8 in 3D. ALEGRA has the capability of running with "1-irregular" mesh, meaning that an element may neighbor another element that has been refined to one higher or lower level. Since this is restricted to a difference of only one level of refinement, neighboring blocks with different levels of initial refinement will have their borders refined such that the 1 level

difference is enforced.

Note: INITIAL REFINEMENT is not available for structured mesh physics options.

DOMAIN keywords controlling initial refinement are given in Table 23.

Table 23: DOMAIN Keywords for Initial Refinement.

Sub-Keyword	Input	Meaning
MAXIMUM LEVELS	int	Specifies the maximum number of INITIAL REFINEMENT levels.
INITIAL REFINEMENT		see Section 5.10.3 on page 132

This input is contained as a subset within the DOMAIN input. Valid input identifies any combination of sidesets, element blocks, or detonation objects for initial refinement. During initialization all elements on a sideset, in an element block, or adjacent to a detonation object are refined to the maximum level of refinement. This capability is particularly useful for increasing the resolution of a given mesh using the original GENESIS file. Including all element blocks in the list of initial refinement objects, increases the mesh resolution by a factor of four in 2D and eight in 3D.

Table 24: INITIAL REFINEMENT Sub-Keywords

Sub-Keyword	Meaning
ALL BLOCK	Specifies that all element blocks will be refined to the highest refinement level.
BLOCK BOUNDARY	Specifies that elements on the boundaries of all element blocks will be refined to the highest refinement level.
BLOCK BOUNDARY, PATRIARCH	Same as block boundary, but refines elements whose patriarch element has a face on the block boundary.
SIDESET int	Refine the sideset to the highest refinement level.
SIDESET int PATRIARCH	Same as above but refines elements whose patriarch element has a face on the sideset.

continued on next page

continued from previous page	
`SIDESET int SPHERE real vector`	Refine the sideset to the highest refinement level while mapping surface nodes to a sphere of radius r centered at position p. In 2D CIRCLE is an alternate keyword here.
`SIDESET int CYLINDER real vector vector`	Refine the sideset to the highest refinement level while mapping surface nodes to a cylinder of radius r with points p1 and p2 lying on the axis of the cylinder. Primarily applicable in 3D.
`SIDESET int RECONSTRUCT`	Reconstructs a local smooth surface through the sideset nodes using approximate surface normals at these nodes.
`BLOCK int`	Refine the element block to the highest refinement level.
`BLOCK int, LEVEL int`	The element block specified will be refined to the refinement level specified. The specified refinement level must be less than the maximum level set by the `MAXIMUM LEVELS` keyword.
`DETONATION POINT int`	Refine the detonation point to the highest refinement level.

Example:

```
domain
   maximum levels = 2
   initial refinement
      sideset 1
      block 5
      detonation point 7
      block 3, level 1
      block 4, level 2
   end
end
```

Limitations:

1. Currently, the only detonation object supported for `INITIAL REFINEMENT` is the detonation point (see Section 8.5.1 on page 177).

2. Nodeset specified boundary conditions cannot be used with initial refinement. Sidesets must be used to specify both Neumann (*e.g.*, pressure) and Dirichlet (*e.g.*, kinematic) boundary conditions. (see Section 7.1 on page 144) Thus, for a 2D cylindrical geometry, the axial `NO DISPLACEMENT` condition must be specified with a sideset, not a nodeset. The data layout of a nodeset inhibits correct refinement.

5.11 Cell Doctor

```
CELL DOCTOR
   [DISCARD int
   [FREQUENCY int]
   [MINIMUM POSITION, vector]
   [MAXIMUM POSITION, vector]
   [TIME RANGE MIN real MAX real]
   [TRIGGER, variable_name, MINIMUM real MAXIMUM real END]
   ...
END
```

The `CELL DOCTOR` keyword provides facilities for modifying the material content of a cell in an ad hoc manner during a simulation. This section describes its most general form, usable from any physics package. A more specialized version of `CELL DOCTOR`, available under `HYDRODYNAMICS`, is described in Section 8.4.2 on page 175.

5.11.1 Discard

The `CELL DOCTOR` keyword provides one subkeyword, `DISCARD`. This keyword has the syntax

```
DISCARD int
   [VERBOSE]
   [FREQUENCY int]
   [MINIMUM POSITION, vector]
   [MAXIMUM POSITION, vector]
   [TIME RANGE MIN real MAX real]
   [TRIGGER, variable name, MINIMUM real MAXIMUM real]
```

```
END
```

The DISCARD subkeyword provides the user with the capability to remove a selected material within a specified geometrical box within a specific time range when its trigger variable falls within the specified range.

The integer field after the DISCARD keyword can be any material id. The FREQUENCY keyword specifies the frequency at which the discard is performed in computational cycles. Thus a FREQUENCY value of 3 means the discard is done every third computational cycle. The MINIMUM POSITION and MAXIMUM POSITION values specify a spatial window within which elements will be checked for satisfying the discard criteria. Note that the location of the center of the element is tested against this criteria. The TIME RANGE gives a temporal window within which the discard criteria are tested. The VERBOSE keyword causes output to be sent to standard out for every test of the criteria and discard. This feature should only be used for debugging purposes. As a default, the MINIMUM POSITION and MAXIMUM POSITION values are both set to zero, so the user MUST set these values to encompass the region of the problem in which the discard is to take place. The TIME RANGE values are defaulted to zero for the minimum and a very large number for the maximum, so the user can neglect setting these values if the discard is desired at all times. This assumes the problem starts at time=0. or some positive time.

The trigger variable may be any scalar variable of the specified material. A listing of all variables for each material can be found in the .out file produced by an ALEGRA run. Examples of permissible trigger variables that are defined for many materials are DENSITY, PRESSURE, or TEMPERATURE. For example, one could simulate the effects of melting in a solid dynamics simulation with the input:

```
cell doctor
   discard 1 $ material 1 discard specification
     minimum position, x = -55., y = -30., x = -110.
     maximum position, x =  55., y =  50., x =    0.
     trigger, temperature, minimum 1600. maximum 1.0e10
   end
end
```

This specifies that material 1 is discarded when it reaches its melting temperature of 1600 K. The material will be checked every cycle for all ele-

ments with centers in the specified box throughout the simulation, since the FREQUENCY and TIME RANGE keywords have not been specified.

The user can enter as many DISCARD keyword blocks as desired. Overlapping discard specifications are or-ed together to determine if a material should be discarded.

Void is used to replace any material that is deleted. The mass and volume of material that is deleted is tracked and is available for output.

5.12 Tracer Points

```
TRACER POINTS
   LAGRANGIAN TRACER int, vector  [NO INTERP] [ROTATE vector]
   EULERIAN TRACER   int, vector  [NO INTERP] [ROTATE vector]
   ALE TRACER        int, vector  [NO INTERP] [ROTATE vector]
END
```

The TRACER POINTS keyword group begins a list of tracer points, which are used to track material or mesh motion in a calculation. As many tracers of any type may be included.

An EULERIAN TRACER never changes its spatial position.

A LAGRANGIAN TRACER is moved with the interpolated velocity at its local position. It thus moves with the local material velocity. The natural mesh coordinates should not change for a LAGRANGIAN TRACER in a Lagrangian mesh.

An ALE TRACER stays with the original natural coordinates of the mesh and is thus a "mesh tracer." An ALE TRACER will be an EULERIAN TRACER in an Eulerian mesh and a LAGRANGIAN TRACER in a Lagrangian mesh and a mesh tracer in an ALE mesh. The ALE TRACER is the most efficient tracer to use. LAGRANGIAN TRACERs in Eulerian meshes and EULERIAN TRACERs in Lagrangian meshes are the most expensive.

An example tracer specification is:

```
tracer points
   lagrangian tracer   1  x -0.009 y 198.345 z 3.3
```

```
   eulerian tracer     2  x  3.45  y 2.65    z 2.0
   ale tracer          3  x -0.009 y 198.345 z 3.3
   eulerian tracer    30  x -0.009 y 198.345 Z 3.3
                          no interp
   lagrangian tracer 300  x  4.0   y 0.0     z 0.0
                          rotate x 0., y=0., z=45.
end
```

The location of the tracer can be modified using the `ROTATE` keyword, which rotates the tracer about the x, y, and z coordinate axes (and in that order). Rotation values about each coordinate axis are interpreted as degrees.

Tracer locations are written to the HISPLT post-processing database [42]. History variables providing tracer locations in the local coordinate system (`PSX-X`, `PSY-Y`, and `PSY-Z`) are described in Section 13.3 on page 298. Currently HISPLT will not recognize an `ALE TRACER`, listing the type of the tracer as `UNKNOWN` in the catalog, but `LAGRANGIAN` on the plot header.

By default, interpolation of the data to the tracer location in the element is performed for nodal variables. If no interpolation is desired, the keyword `NO INTERPOLATION` can be appended after the tracer coordinates for each tracer point. In that case, the value at the nearest node is written to the database although the actual position of the tracer behaves as lagrangian, eulerian, or ale. Interpolation is not yet available for element variables, so the value of an element variable written to the database is simply the value at the element center.

For the structured mesh option, tracers are fully functional, but the element id that is returned is not easily interpreted since it is an actual index into the block element array. The nearest node function for the `NO INTERP` option is not functioning for structured mesh physics options.

5.12.1 Eulerian Tracer

```
[EULERIAN TRACER int vector [NO INTERP] [ROTATE, vector]]
```

In two dimensions, use `[ROTATE, Z real]`. In three dimensions, specify the angle of rotation in degrees about each axis, such as

```
ROTATE, X 30. Y -10. Z 45.
```

5.12.2 Lagrangian Tracer

```
[LAGRANGIAN TRACER int vector [NO INTERP] [ROTATE, vector]]
```

In two dimensions, use `[ROTATE, Z real]`. In three dimensions, specify the angle of rotation in degrees about each axis.

5.12.3 ALE Tracer

```
[ALE TRACER int vector [NO INTERP] [ROTATE, vector]]
```

In two dimensions, use `[ROTATE, Z real]`. In three dimensions, specify the angle of rotation in degrees about each axis.

5.13 Functions

```
FUNCTION int
  real real
  real real
  ... ...
END
```

The `FUNCTION` keyword group defines a function in terms of a table of real ordered pairs. A function is identified by the `int` integer field after the `FUNCTION` keyword. The integer field lets the `FUNCTION` be referenced by a `function-set` as part of other keywords that define the problem, and so may be used to specify such things as boundary conditions as functions of time or space. Linear interpolation is used between table points. To approximate steps in a function, the user should use x-values that are close together. For example:

```
FUNCTION 11
  0.0    0.0
  1.0    0.0
  1.0001 1.0
  2.0    1.0
END
```

ALEGRA provides a predefined constant function with id 0, equivalent to

```
FUNCTION 0
  -REAL_MAX/2.  1.
   REAL_MAX/2.  1.
END
```

that allows a constant function to be referenced conveniently in the code input. Here "`REAL_MAX`" is the largest floating point number available to the code. Since `FUNCTION 0` has been predefined, the user cannot use 0 as a function identifier when defining additional functions.

If automatic `UNITS` conversion is desired (as described in Section 4.1.3 on page 69), *the unit conversion string must be entered after each applicable real value in the function.*

6 Energetics Input

```
{physics choice keyword}
   ...
   [ENERGETICS]
      ...
      [energetics keywords]
      ...
   [END]
   ...
END
```

The `physics choice keyword` is one of the options described in Section 5.2 or 5.3 on pages 93 and 94. The `ENERGETICS` keyword and its corresponding `END` keyword are optional and may be included in the input file for clarity.

6.1 Energetics I/O Control

6.1.1 Detailed Energy Tallies

```
DETAILED ENERGY TALLIES
```

In many problems of interest it is often desirable to know the energy budget. How much energy is related to a given physical process? What is the value of the kinetic and internal energies? How much energy is supplied by a given source or is lost to a given sink? How fast does energy change from one form to another?

ALEGRA provides the user with a detailed set of energy and power tallies to answer such questions. The tallies are written to both the HISPLT and EXODUS output files. Typically the HISPLT file will contain tallies at more frequent intervals compared to the EXODUS file (depending on the user specification – see the `EMIT HISPLT` and `EMIT PLOT` commands) because it is smaller and does not contain mesh information or plot variables.

The Tables 91 and 92 in Chapter 13 summarize the various energy and power tallies that are available. The tallies are grouped by choice of the

physics specification keyword, described in Sections 5.2 and 5.3. Extra global power tallies marked with an asterisk (*) are omitted from the output files unless the DETAILED ENERGY TALLIES keyword is specified. The DETAILED ENERGY TALLIES keyword causes extra global power tallies to be included in the history and plot dump files.

6.2 Energy Sources

6.2.1 Energy Deposition

```
{EULERIAN | LAGRANGIAN} ENERGY DEPOSITION, block-id
   TIME function-set
   [MATERIAL material-id]
   [RADIAL function-set]
   [CYLINDRICAL]
   [MASS WEIGHT]
   [VOLUME WEIGHT]
END
```

This keyword group causes energy to be deposited in the specified block over time. The energy can be optionally restricted to one material in this block by the use of MATERIAL keyword. Use the EULERIAN prefix for Eulerian-mesh blocks and the LAGRANGIAN prefix for Lagrangian-mesh blocks.

The power distribution is given as a separable function of radius from the origin and time, and it should be in dimensions of energy per second. If the CYLINDRICAL keyword is included, the RADIAL value is interpreted to be the cylindrical radius, otherwise it is interpreted as the spherical radius. The keywords MASS WEIGHT and VOLUME WEIGHT control how the energy is apportioned among the materials in a multi-material block.

The function-sets used here support the SHIFT parameter described in 3.2.5, which allows the user to choose the zero time.

The user specifies an energy deposition rate as a function of time and space. At every cycle within the time range of this table, ALEGRA will compute the total mass or volume of the elements targeted within the block. The radial weight function specified by the RADIAL keyword will be included also. The choice of mass or volume is controlled by the MASS WEIGHT or

VOLUME WEIGHT keywords. If the MATERIAL keyword is specified, then the mass or volume of only that material within the elements of the block will be taken into account. This quantity is then used to normalize the user specified rate term so that an energy/mass/time or energy/volume/time rate is obtained. This specific rate is then applied to the element or the specified material within the element. In this way, movement of material into or out of the block will not affect the total energy deposited into the block over the time span of the table.

The user should restrict the TIME span of the table to only the time that the energy source term is desired. Putting a larger time span in with zero rate values causes the code to compute normalization factors needlessly.

7 Mechanics Input

```
{physics choice keyword}
   ...
   [MECHANICS]
      ...
      [mechanics keywords]
      ...
   [END]
   ...
END
```

Mechanics algorithms deal with motion of material in space. The only released model for this type of algorithm is a dynamically evolving distribution of matter, modeled with explicit advancement of the variables at every step in time. Another class of Mechanics algorithms being developed for ALEGRA are statics models, where an implicit scheme is used to determine the final state of some material distribution subject to a set of internal and external forces.

The `physics choice keyword` is one of the options described in Section 5.2 or 5.3. The `MECHANICS` keyword and its corresponding `END` keyword are optional and may be included in the input file for clarity.

7.1 Boundary Conditions and Body Forces

In the following boundary condition descriptions, the mesh portion to which the constraint is applied is often described as either a nodeset or a sideset, *i.e.*, {`nodeset | sideset`}. However, when adaptivity is being used, only sidesets should be used to specify constraints in the region of the mesh in which adaptivity may be changing element connectivity and adding elements. This is because ALEGRA currently does NOT support refined nodesets and thus a constraint specified on a nodeset will NOT be applied to the newly added nodes on that mesh boundary. If the boundary section is not in a region of mesh where adaptivity may be changing the mesh, nodesets can still be used to specify constraints.

7.1.1 Gravity

```
GRAVITY vector
```

This keyword specifies that the force of gravity will be applied in a calculation in the direction specified. The gravitational potential energy is output as an energy diagnostic. Normally gravity is not included.

7.1.2 No Displacement

```
NO DISPLACEMENT, {nodeset | sideset}
   { X | Y | Z | R | RADIAL | NORMAL | TANGENT} [vector] }
```

This keyword allows the user to hold one or more coordinates of a set of nodes fixed. If `X`, `Y`, `Z`, or `R` are specified, then `vector` should not be entered. Multiple `NO DISPLACEMENT {X | Y | Z | R}` boundary conditions may be specified for the same set of nodes to constrain motion in more than one direction. Note that in 3D or 2D Cartesian geometry, `X`, `Y` and `Z` refer to their standard components. In 2D cylindrical geometry, `R` and `Z` refer to the radial and axial components, respectively.

If `RADIAL` is specified, then `vector` must be entered. It is assumed that `vector` specifies the center through which the constraining force acts. Nodes are constrained to not move along the line connecting the node with `vector`.

If `NORMAL` or `TANGENT` is specified, then `vector` must be entered. It is assumed that the nodes are coplanar (collinear in 2D) and that `vector` specifies the outward boundary normal. Nodes are then constrained to not move in either the normal or tangential direction.

Note that if a node is constrained by both a `{NORMAL | TANGENT}` boundary condition and an `{X | Y | Z | R}` boundary condition and the boundary normals are not orthogonal, then application of the `{NORMAL | TANGENT}` constraint will add an acceleration component along `{X | Y | Z | R}`. In this case, the common node should be placed in its own `nodeset` and multiple `{X | Y | Z | R}` constraints should be specified. A related problem exists for the shared nodes along the common edge between two non-orthogonal no displacement surfaces. A complete description is provided on page 308.

All nodes from the mesh database which are part of the specified `nodeset` or `sideset` will be subject to the boundary condition.

Structured mesh physics options support only nodesets for the application of `NO DISPLACEMENT` boundary conditions.

7.1.3 No Cylindrical Displacement

```
NO CYLINDRICAL DISPLACEMENT, {nodeset | sideset}
   { RADIAL | CIRCUMFERENTIAL | AXIAL }
```

This keyword allows the user to hold one or more coordinates of a set of nodes fixed in the `RADIAL`, `CIRCUMFERENTIAL`, or `AXIAL` directions. Multiple `NO CYLINDRICAL DISPLACEMENT` boundary conditions may be specified for the same set of nodes to constrain motion in more than one direction.

Structured mesh physics options support only nodesets for the application of `NO CYLINDRICAL DISPLACEMENT` boundary conditions.

7.1.4 Prescribed Force

```
PRESCRIBED FORCE, nodeset, direction-function
```

This keyword allows the user to prescribe a force acting on a nodeset as a function of time in the direction specified by the `direction-function` construct as described in Section 3.2.9 on page 67.

7.1.5 Rigid Segment

```
RIGID SEGMENT, nodeset
   ENDPOINT1, vector
   ENDPOINT2, vector
END
```

This keyword group allows the user to specify a rigid segment in two dimensions or, in three dimensions, a rigid plane that extends infinitely in

the z direction parallel to the segment that cannot be penetrated by the nodes of the nodeset. In two dimensions, a series of rigid segments allows for the creation of a rigid body.

7.1.6 Rigid Surface

```
RIGID SURFACE, {nodeset | sideset}
   CENTER, vector
   NORMAL, vector
   STATIC COEF   real
   VELOCITY COEF real
   DECAY COEF    real
END
```

In three dimensions, this keyword group allows the user to specify a rigid, plane surface (not part of the database mesh) that cannot be penetrated by the (slave) surface specified by the nodeset or sideset.

7.1.7 Traction BC

```
TRACTION BC, sideset symtensor function-set
```

This keyword allows the user to specify a stress given by symtensor that acts on a surface. The stress is "dotted" with the unit normal of the element's surface to give a force that is proportioned and applied to the nodes that lie on the surface.

7.2 Mechanics Algorithm Control

7.2.1 Hourglass Control

```
{PISCES | PRONTO} HOURGLASS CONTROL [int]
   [parameter real]
END
```

Hourglass stiffening is added to a problem to couple the element hourglass modes to the element internal energy. This prevents runaway distortion from uncoupled, uncontrolled hourglass modes.

Two models for hourglass control are available: `PISCES`, and `PRONTO`, as described in the following subsections. The optional integer identifier after the `HOURGLASS CONTROL` keyword identifies an instance of the model that can be applied selectively to different blocks in the problem (see the `BLOCK` keyword).

Each model uses various parameter keywords to specify viscosity and/or stiffness values appropriate to the model.

Pisces Hourglass Control

```
PISCES HOURGLASS CONTROL [int]
   [VISCOSITY real (0.05)]
END
```

The bulk viscosity defined by the `VISCOSITY` keyword must be less than 0.25 to assure stability.

Pronto Hourglass Control

```
PRONTO HOURGLASS CONTROL [int]
   [VISCOSITY real (cartesian: 0.00) (cylindrical: 0.03)]
   [STIFFNESS real (cartesian: 0.05) (cylindrical: 0.01)]
END
```

The stiffness coefficient is the most effective since it is adds a permanent restoring force rather than a viscous correction. What this does to the accuracy of the calculation is unclear. The `PRONTO` name is a misnomer since the coding attempts to follow Reference [12] rather than what might be implemented in PRONTO [38, 39].

7.2.2 Moving Mesh

```
MOVING MESH vector
```

This keyword causes all Eulerian or smoothed Eulerian regions of the computational mesh to move through space at a prescribed velocity. The velocity of the material contained in the mesh is not prescribed. This keyword can be used to keep the mesh centered on the region of interest when the velocity of the region of interest is known a priori. The `BLOCK` input subkeyword `SMOOTHED EULERIAN MESH` may be helpful to smooth mesh irregularities with this option.

7.2.3 Track

```
TRACK, block-id
```

When this keyword is in effect, ALEGRA performs the calculation in a reference frame that is at rest with respect to the specified block. The mean velocity of the block is computed at the end of each time step, and this velocity is subtracted from the velocity of all nodes in the simulation. ALEGRA keeps track of the cumulative effects of each change in reference frame and corrects its plot output to be in the original reference frame; hence, this option is transparent to post-processing tools. The main effect of the `TRACK` option is to cause the computation mesh in Eulerian and smoothed Eulerian regions to move at roughly the same velocity as the tracked block, so as to follow the block. The `TRACK` feature is not supported for structured mesh physics options.

8 (Hydro)Dynamics Input

```
HYDRODYNAMICS
   ...
   [hydrodynamics keywords]
   ...
   [DYNAMICS]
      ...
      [dynamics keywords]
      ...
   [END]
   ...
END
```

HYDRODYNAMICS is one of the options described in Sections 5.2 and 5.3. The DYNAMICS sub-keyword and its corresponding END keyword are optional and may be included in the input file for clarity.

8.1 Dynamics Initial Conditions

8.1.1 Initial Velocity

```
INITIAL VELOCITY, nodeset, vector
```

This keyword allows the user to specify an initial velocity for each node in a nodeset. This input will take precedence over any values set by INITIAL BLOCK VELOCITY or INITIAL ANGULAR VELOCITY input. If the same node is in multiple nodesets then the last specified value will be used.

8.1.2 Initial Angular Velocity

```
INITIAL ANGULAR VELOCITY, {block-id | nodeset},
   CENTER, vector, VALUE, {real | vector}
```

This keyword allows the user to specify an initial angular velocity (radians/second) for each node of a block or a nodeset. The CENTER keyword

specifies a point on the axis of rotation. The VALUE keyword specifies the angular velocity. In 2D Cartesian geometry, the real magnitude of the angular velocity is specified, as the axis of rotation is assumed to be orthogonal to the mesh, i.e., along the positive z axis. In 3D the full angular velocity vector is specified, i.e., the direction of the vector specifies the axis of rotation and the magnitude of the vector specifies the rate of rotation. The initial velocity of a node is the cross product of the angular velocity with the distance of the node from the center point.

For example,

```
INITIAL ANGULAR VELOCITY, NODESET 1
CENTER, X 0.0 Y 0.0 Z 0.0
VALUE, X 0.0 Y 0.0 Z 6.283185E2
```

specifies 100 revolutions per second about the z axis.

The initial angular velocity keyword is not supported for 2D axisymmetric geometry.

8.1.3 Initial Block Velocity

```
{INITIAL BLOCK VELOCITY, block-id,
   { {X | Y | Z | R} real |
     VECTOR, vector       |
     {RADIAL | NORMAL | TANGENT}, vector real }
```

This keyword allows the user to specify an initial velocity for each node of a block. Three different forms of this keyword are available, as shown above.

In the first form, a principal direction (X, Y, or Z in Cartesian geometry or R or Z in 2D cylindrical geometry) is specified, followed by the magnitude of the velocity in that direction. For example,

```
initial block velocity, block 3, x 1.0e4
```

will set the x-component of the velocity for each node in block 3 to 1.0e4.

In the second form, a velocity `vector` is specified after the `VECTOR` keyword. For example,

```
initial block velocity, block 3, vector,
                        x 1.0e4 y 2.0e4, z 1.0e4
```

will set the velocity of each node in block 3 to (1.0e4, 2.0e4, 1.0e4).

In the third form, the `RADIAL` keyword specifies that the velocity vector lies along the line from the point given by `vector` to each node in the block. The `TANGENT` keyword (2D only), on the other hand, specifies that the velocity vector is tangent to the vector from each node to the point given by `vector`. In either case, the velocity vector is normalized and multiplied by a velocity magnitude given by `real`. For example,

```
initial block velocity, block 3, radial, x 0. y 0. z 0., -1.0e8
```

will initialize each node in block 3 to a velocity of magnitude 1.0e8 moving toward the origin.

Finally, the `NORMAL` keyword specifies a vector from the origin to the point given by vector. This vector is NOT normalized, and it is multiplied by the factor given by `real` and applied to each node in the block. For example,

```
initial block velocity, block 3, normal,
                        x 1.0 y 2.0 z 1.0, 1.0e4
```

is equivalent to the previous example using the `VECTOR` keyword.

If the same block is specified more than once or if a node is in multiple block-ids then the last specified value will be used.

8.1.4 Random Block Velocity

```
RANDOM BLOCK VELOCITY, block-id, VECTOR, vector
```

A random velocity will be generated for each non-zero vector component with a magnitude ranging between +/- of the specified vector component.

```
RANDOM BLOCK VELOCITY, BLOCK 1, VECTOR, X 2.0 Y 0.0 Z 0.0
```

will generate x-velocities ranging from -2 to 2.

8.1.5 Sinusoid Velocity

```
SINUSOID VELOCITY, nodeset, SCALE vector, LAMBDA real,
   {X | Y | Z | R} real
```

This keyword allows the user to specify an initial sinusoidal velocity perturbation for a given nodeset. `SCALE` ($\vec{v}_0$) specifies the absolute amplitude and direction of the perturbation, `LAMBDA` (λ) specifies the wavelength, and `X`, `Y`, `Z`, or `R` (x_0) specifies both the direction and the phase. `R` is restricted to the radial coordinate in 2D cylindrical simulations. Perturbations are calculated according to the formula:

$$\vec{v} = \vec{v}_0 \cdot \cos\left(\frac{2\pi}{\lambda}(x - x_0)\right) \tag{8.2}$$

Example:

```
sinusoid velocity, nodeset 4, scale, x 0.001 y 0. lambda 1. y 0.
```

8.2 Dynamics Initial Density and Surface Perturbations

8.2.1 Cylindrical Mode Density

```
CYLINDRICAL MODE DENSITY, block-id, RANGE real,
   [ANGULAR WAVENUMBER int,] [THETA real,]
   [WAVELENGTH real,] [RADIAL WAVELENGTH real,] [RADIAL PHASE real,]
   {X | Y | Z} real
```

This keyword allows the user to specify an initial `CYLINDRICAL MODE DENSITY` perturbation for a given block. This keyword does not apply to 2D cylindrical geometry.

All parameters default to the value 0.0 if omitted. Except for the `block-id`, they may be listed in any order. Axial coordinates are specified in the native problem units (e.g., m or cm), while angles are specified in degrees. The block should be cylindrical in shape and centered on one of the principal coordinate axes. `RANGE` specifies the relative amplitude ϵ of the perturbation.

The `ANGULAR WAVENUMBER`, N, specifies the number of azimuthal periods the perturbation has about the axis and `THETA` specifies the angular phase θ_0. If the wavenumber is zero, the perturbation will be rotationally uniform.

The `WAVELENGTH` specifies the axial wavelength λ. If the wavelength is zero, the perturbation will be axially uniform. If the wavelength and the wavenumber are both nonzero, the perturbation will be helical. The direction of rotation of the helix can be controlled by the sign of the wavelength.

The `RADIAL WAVELENGTH` specifies the radial wavelength λ_r. The `RADIAL PHASE` specifies the radial phase r_0. If the radial wavelength is zero, the perturbation will be radially uniform. The radius is computed relative to the cylindrical axis.

One of `X`, `Y`, or `Z` specifies both the coordinate axis that coincides with the cylinder axis and the axial phase z_0. (`Z` is permissible in 2D Cartesian geometry.) Perturbations are calculated according to the formula:

$$\rho = \rho_0 \cdot \left(1 + \epsilon \cdot \cos\left(\frac{2\pi}{\lambda}(z - z_0) + \frac{2\pi}{\lambda_r}(r - r_0) + N(\theta - \theta_0)\right)\right) \qquad (8.3)$$

3D Cartesian example:

```
cylindrical mode density, block 1,
   range       =  0.5
   ang wavenum =  3
   theta       =  0.   $ any multiple of 120. produces the same
   wavelength  = -1.
   z           = 0.
```

2D Cartesian example:

```
cylindrical mode density, block 1,
```

```
   range     = 1.
   rad wave  = 1.
   rad phase = 0.5
   z         = 0.    $ direction of axis
```

8.2.2 Cylindrical Mode Surface

```
CYLINDRICAL MODE SURFACE, nodeset, RANGE real,
   ANGULAR WAVENUMBER int, [THETA real,]
   [WAVELENGTH real,] [RADIAL WAVELENGTH real,] [RADIAL PHASE real,]
   {X | Y | Z} real
```

This keyword allows the user to specify an initial cylindrical mode surface perturbation for a given nodeset. This keyword does not apply to 2D cylindrical geometry.

All parameters default to the value 0.0 if omitted. Except for the `nodeset`, they may be listed in any order. Radial and axial coordinates are specified in the native problem units (e.g., m or cm), while angles are specified in degrees.

The original `nodeset` should be cylindrical in shape and centered on one of the principal coordinate axes. `RANGE` specifies the absolute amplitude ϵ of the perturbation. The direction of the perturbation is orthogonal to the cylinder's axis. The amplitude should be less than the width of a mesh element that abuts the surface, otherwise inverted elements may be generated.

The `ANGULAR WAVENUMBER` specifies the number of azimuthal periods N the perturbation has about the axis and `THETA` specifies the angular phase θ_0. If the wavenumber is zero, the perturbation will be rotationally uniform.

The `WAVELENGTH` specifies the axial wavelength λ. If the wavelength is zero, the perturbation will be axially uniform. If the wavelength and the wavenumber are both nonzero, the perturbation will be helical. The direction of rotation of the helix can be controlled by the sign of the wavelength.

The `RADIAL WAVELENGTH` specifies the radial wavelength λ_r. The `RADIAL PHASE` specifies the radial phase r_0. If the radial wavelength is zero, the perturbation will be radially uniform. The radius is computed relative to the cylindrical axis.

One of `X`, `Y`, or `Z` specifies both the coordinate axis that coincides with the cylinder axis and the axial phase z_0. (`Z` is permissible in 2D Cartesian geometry.) Perturbations are calculated according to the formula of the form:

$$\vec{x} = \vec{x}_0 \cdot \left(1 + \frac{\epsilon}{|\vec{x}_0|} \cdot \cos\left(\frac{2\pi}{\lambda}(z - z_0) + \frac{2\pi}{\lambda_r}(r - r_0) + N(\theta - \theta_0)\right)\right) \quad (8.4)$$

3D Cartesian example shown in Figure 10:

```
cylindrical mode surface, nodeset 1,
   range       = 0.1
   ang wavenum = 4
   theta       = 0.   $ any multiple of 90. produces the same
   length      = 1.
   z           = 0.
```

2D Cartesian example shown in Figure 12:

```
cylindrical mode surface, nodeset 1,
   ang wavenum = 12
   theta       = 0.    $ any multiple of 30. produces the same
   range       = 0.02
   z           = 0.
```

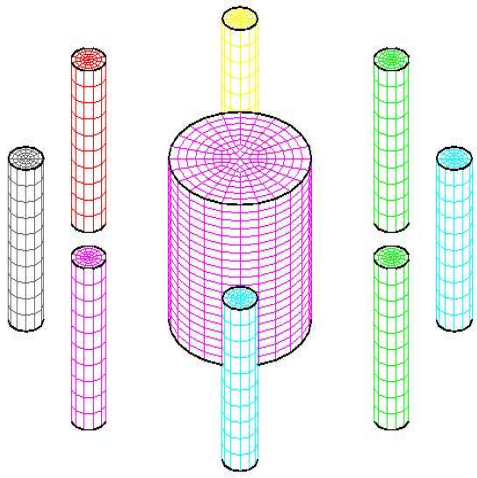

Figure 9: Initial 3D Cartesian geometry.

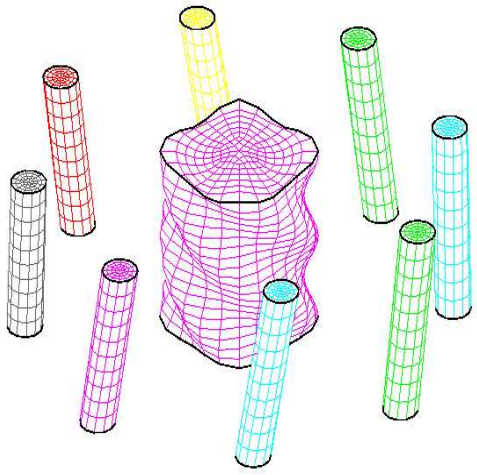

Figure 10: Perturbed 3D Cartesian geometry. A `CYLINDRICAL MODE SURFACE` perturbation is applied to the lateral surface of the central cylinder, while a `SINUSOID SURFACE` perturbation is applied to its top surface. The outer ring of cylinders shows the effect of a `TWISTED MESH` perturbation.

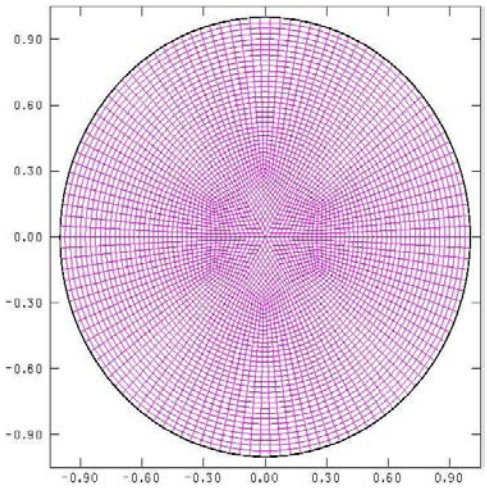

Figure 11: Initial 2D Cartesian geometry.

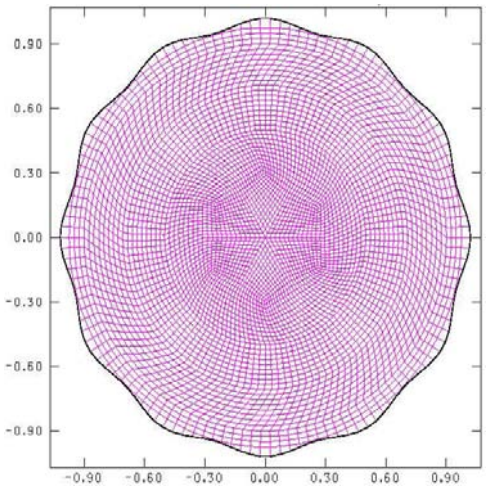

Figure 12: Perturbed 2D Cartesian geometry. A `CYLINDRICAL MODE SURFACE` perturbation is applied to the outer boundary of the circular mesh, while the central portion shows the effect of a `TWISTED MESH` perturbation.

8.2.3 Degenerate Surface

```
DEGENERATE SURFACE, nodeset, POINT, vector       (2D or 3D)

DEGENERATE SURFACE, nodeset, AXIS, vector1 vector2,
   TOLERANCE real (1e-12)                        (3D only)
```

This keyword allows the user to specify a degenerate surface for a given `nodeset`. A degenerate surface collapses all nodes down to a single point or a set of points along a line. The primary intent for this mesh modification is to remove "holes" in cylindrically or spherically symmetric meshes, therefore the user should construct a mesh that is a close approximation to the desired geometry. Even though many nodes appear to be one point, all points in the original `nodeset` are retained.

The `POINT` subkeyword collapses all nodes in the `nodeset` to the point specified by `vector`. This option is available in both 2D and 3D.

The `AXIS` subkeyword collapses all nodes in the `nodeset` to the line defined by `vector1` and `vector2`. `vector1` specifies a point on the line and `vector2` specifies the direction of the line and need not be normalized. The algorithm computes the perpendicular projection of a node to the axis. The `TOLERANCE` keyword is a matching criteria for grouping points along the axis into sets of degenerate points. Points that are separated by a distance less than the tolerance are treated as a single degenerate point. Points that are separated by a distance greater than the tolerance belong to different degenerate points. This option is available only in 3D since it makes no sense for 2D.

`DEGENERATE SURFACE` may be used in conjunction with other boundary conditions such as `DEGENERATE BC`, `NO DISPLACEMENT`, or `PRESCRIBED VELOCITY` to produce a variety of desired results.

3D example projecting a spherical cutout around the origin to the origin:

```
degenerate surface, nodeset 1, point, x 0. y 0. z 0.
```

3D example projecting a cylindrical cutout along the z-axis to that axis:

```
degenerate surface, nodeset 1, axis, x 0. y 0. z 0.,
```

```
x 0. y 0. z 1.,   tol 1.e-9
```

2D cylindrical example projecting a circular cutout around the origin to the origin:

```
degenerate surface, nodeset 1, point, r 0. z 0.
```

8.2.4 Random Density

```
RANDOM DENSITY, block-id, RANGE real, [SEED integer(1)],
               [AGGREGATE SIZE real]
```

This keyword allows the user to specify an initial random density perturbation for a given block. RANGE specifies the relative amplitude ϵ of the perturbation. Perturbations are calculated according to the formula:

$$\rho = \rho_0 \cdot (1 + \epsilon(2 \cdot \text{random}() - 1)) \tag{8.5}$$

where the random number is between zero and 1.

Example:

```
random density, block 8, range 0.01, seed 2342345,
                aggregate size .001
```

specifies a 1% random density perturbation with a seed and an aggregate size of .001. The random number generator SEED can be given as an integer and default to 1. The AGGREGATE SIZE specifies the approximate size of aggregates computed using a Hilbert space filling curve based algorithm. If no aggregate size is given then the block is fit inside a square box with an estimated 100 aggregates in each direction. Generally the user will want to request an aggregate size. Requesting very small aggregate sizes may result in a noticeable extended setup time but will in general be noticeable for normal meshes with aggregate sizes on the order of or greater than then smallest element dimension. The aggregate sizes will be roughly independent of the number of elements or the mesh stretching. If the aggregate size chosen is much smaller than the dimension of the smallest element, each element will

have an density uncorrelated with its element neighbors. This algorithm is independent of the number of processors utilized.

8.2.5 Random Surface

```
RANDOM SURFACE, nodeset, RANGE vector
```

This keyword allows the user to specify an initial random surface perturbation for a given nodeset. `RANGE` specifies the absolute amplitude of the perturbation components. Each coordinate for each node of the nodeset is randomly perturbed according to its respective amplitude component according to the formula:

$$\vec{x} = \vec{x}_0 + \vec{x}_{range} \cdot (2 \cdot \text{random}() - 1)) \tag{8.6}$$

and the random number is between zero and one.

3D Cartesian example:

```
random surface, nodeset 29, range, x 0.1 y 0.02 z 0.005
```

8.2.6 Sinusoid Density

```
SINUSOID DENSITY, block-id, RANGE real, WAVELENGTH real,
   {X | Y | Z | R} real
```

This keyword allows the user to specify an initial sinusoidal density perturbation for a given block. `RANGE` specifies the relative amplitude ϵ of the perturbation, `WAVELENGTH` specifies the wavelength λ, and `X`, `Y`, `Z`, or `R` specifies both the coordinate direction and the phase x_0. `R` is restricted to the radial coordinate in 2D cylindrical simulations. Perturbations are calculated according to the formula:

$$\rho = \rho_0 \cdot \left(1 + \epsilon \cdot \cos\left(\frac{2\pi}{\lambda}(x - x_0)\right)\right) \tag{8.7}$$

Perturbations in multiple directions are defined by repeated use of this keyword.

Example:

```
sinusoid density, block 8, range .01, wavelength 1., z 0.
```

Special capability for structured grids ONLY: The keyword `MEAN` after `SINUSOID DENSITY` requests that the average density be applied in each cell rather than the midpoint value. This is useful if mean initial values are required rather than point values.

Example:

```
sinusoid density, mean, block 8, range .01, wavelength 1., z 0.
```

8.2.7 Sinusoid Surface

```
SINUSOID SURFACE, nodeset, RANGE vector, WAVELENGTH real,
   {X | Y | Z | R} real
```

This keyword allows the user to specify an initial sinusoidal surface perturbation for a given `nodeset`. `RANGE` specifies the absolute amplitude $\vec{x}_{range}$ of the perturbation, `WAVELENGTH` specifies the wavelength λ, and `X`, `Y`, `Z`, or `R` specifies both the coordinate direction and the phase x_0. `R` is restricted to the radial coordinate in 2D cylindrical simulations. Perturbations are calculated according to the formula:

$$\vec{x} = \vec{x}_0 + \vec{x}_{range} \cdot \cos\left(\frac{2\pi}{\lambda}(x - x_0)\right) \tag{8.8}$$

Perturbations in multiple directions are defined by repeated use of this keyword.

3D Cartesian example shown in Figure 10:

```
sinusoid surface, nodeset 1,
```

```
   range, x 0.0 y 0.0 z 0.02,
   wavelength, 1.,
   x 0.
```

2D Cartesian example:

```
sinusoid surface, nodeset 5,
   range = x 0.1,
   length = 1.0,
   y = 0.
```

2D cylindrical example:

```
sinusoid surface, nodeset 5,
   range = x 0.05 y 0.,
   length = 1.0,
   z = -0.25
```

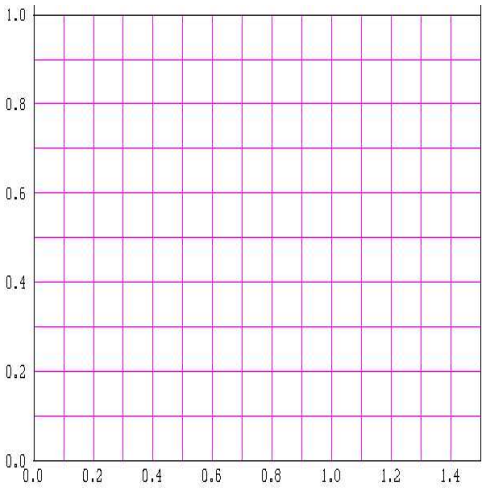

Figure 13: Initial perturbations in 2D cylindrical geometry.

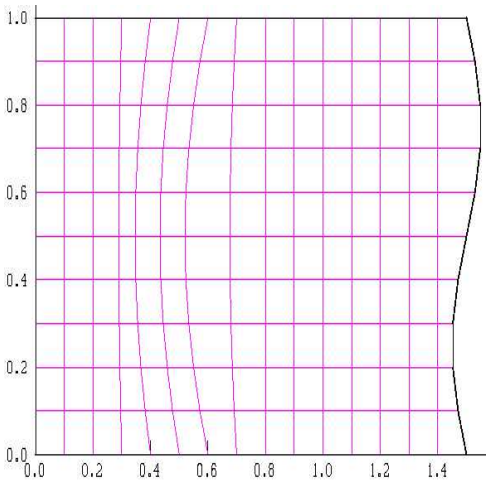

Figure 14: Final perturbations in 2D cylindrical geometry. A `SINUSOID SURFACE` perturbation is applied to the outer radius of the mesh, while the interior hyperbolic curves show the effect of a `TWISTED MESH` perturbation.

8.2.8 Twisted Mesh

```
TWISTED MESH, [THETA0 real], [THETA1 real], [Z0 real],
   [Z1 real], [R0 real], [R1 real], [R2 real], [R3 real]
```

This keyword allows the user to twist the mesh around the z axis in 2D or 3D. All parameters default to the value 0.0 if omitted. They may be listed in any order. Radial and axial coordinates are specified in the native problem units (m or cm), while angles are specified in degrees.

The mesh is rotated by the angle θ_0 when $z \leq Z_0$ and by the angle θ_1 when $Z_1 \leq$z . When $Z_0 \leq z \leq Z_1$, the point is projected to the $z = Z_0$ and $z = Z_1$ planes, the projected points are rotated, and the new coordinates are found by linear interpolation. Thus lines in the unmodified mesh transform to lines in the twisted mesh (perhaps with kinks at Z_0 and Z_1).

If only a radial annulus of the mesh between $R_1 \leq r \leq R_2$ is to be twisted, then the angles of rotation are adjusted to rise linearly from zero between R_0 and R_1, and then return linearly to zero between R_2 and R_3, providing smooth transition regions.

It is possible to divide the total angular twist equally between a counter rotation at Z_0 and a rotation at Z_1, *i.e.*, let $\theta_0 = -\theta/2$ and $\theta_1 = \theta/2$. The following two examples actually produce similar results:

3D Cartesian examples shown in Figure 10:

```
twisted mesh, theta 15.0,
              r0 2., r1 2.5, r2 3.5, r3 4.0
              z0 -2., z1 2.

twisted mesh, theta0 -7.5,
              theta1 7.5,
              r0 2., r1 2.5, r2 3.5, r3 4.0
              z0 -2., z1 2.
```

For 2D Cartesian meshes, θ_0, Z_0 and Z_1 are ignored, and all rotations are based upon the angle θ_1 (in this case the parameter name `THETA` will match `THETA1`). Radial parameters still apply.

2D Cartesian example:

```
twisted mesh, theta 10.0
              r0 0.3, r1 0.5, r2 0.7, r3 0.9
```

For 2D cylindrical meshes, this command will transform straight lines parallel to the z axis into hyperbola of one sheet (and other curves accordingly), because the rotations are computed in 3D and then converted back to the RZ plane. The following two examples actually produce the same results:

2D RZ examples:

```
twisted mesh, theta0 -5.0, theta1 5.0
              r0 0.2, r1 0.4, r2 0.6, r3 0.8
              z0 0.0, z1 1.0


twisted mesh, theta 10.0,
              r0 0.2, r1 0.4, r2 0.6, r3 0.8
              z0 0.0, z1 1.0
```

8.3 Dynamic Boundary Conditions

8.3.1 Degenerate BC

```
DEGENERATE BC, nodeset, POINT, vector          (2D or 3D)

DEGENERATE BC, nodeset, AXIS, vector1 vector2,
   TOLERANCE real (1e-12)                      (3D only)
```

This keyword allows the user to specify a degenerate boundary condition for a given `nodeset`. A degenerate boundary condition constrains all nodes belonging to a degenerate point to move at the same center-of-mass velocity. `DEGENERATE BC` is intended to be used in conjunction with the `DEGENERATE SURFACE` surface perturbation to keep degenerate nodes together.

The `POINT` and `AXIS` subkeywords are for compatibility with the `DEGENERATE SURFACE` keyword and are not used. The `TOLERANCE` keyword is a matching criteria for grouping points along the axis into sets of degenerate points. Points that are separated by a distance less than the tolerance are treated

as a single degenerate point. Points that are separated by a distance greater than the tolerance belong to different degenerate points.

`DEGENERATE BC` may be used in conjunction with other boundary conditions such as `NO DISPLACEMENT` or `PRESCRIBED VELOCITY` to produce a variety of desired results.

3D example projecting a spherical cutout around the origin to the origin:

```
degenerate bc, nodeset 1, point, x 0. y 0. z 0.
```

3D example projecting a cylindrical cutout along the z-axis to that axis:

```
degenerate bc, nodeset 1, axis, x 0. y 0. z 0.,
                                x 0. y 0. z 1.,
                                tol 1.e-9
```

2D cylindrical example projecting a circular cutout around the origin to the origin:

```
degenerate bc, nodeset 1, point, r 0. z 0.
```

8.3.2 Prescribed Acceleration

```
PRESCRIBED ACCELERATION, nodeset, {X | Y | Z}, function-set
```

This keyword allows the user to prescribe an acceleration for a `nodeset` in the direction indicated as a function of time.

8.3.3 Prescribed Velocity

```
PRESCRIBED VELOCITY, {nodeset | sideset}, direction-function
```

This keyword allows the user to prescribe the velocity of a `nodeset` or `sideset` as a function of time as specified by the `direction-function` keyword.

Example:

```
prescribed velocity, nodeset 1, y, function 1

function 1
   0.0 1000.0E5
   1.0 1000.0E5
end
```

8.3.4 Pressure BC

```
PRESSURE BC, {sideset function-set |
              sideset [MOMENT int function-set]
                      [MOMENT int function-set] ... }
END
```

This keyword allows the user to specify a spatially constant pressure that acts on a surface as a function of time. Alternatively, a keyword group may be used to specify the pressure over the surface as a set of spherical harmonic moments.

8.3.5 Pressure Wave

```
PRESSURE WAVE, sideset, vector, PEAK, function-set, RISE,
               function-set, PROPAGATION SPEED real,
               ARRIVAL TIME real
```

This keyword allows the user to simulate a shock wave resulting from a disturbance at a single point given by the `vector` keyword. The two `function-sets` describe the values of the `PEAK` and `RISE` as a function of the distance from the origin point. In the absence of an `ARRIVAL TIME` the time to arrive at a node in the `sideset` is equal to distance from the origin point to the node divided by the `PROPAGATION SPEED`. Otherwise, the time until the wave influences a node is the sum of the `ARRIVAL TIME` and the distance divided by the `PROPAGATION SPEED`. The pressure imposed on the surface is described by the following equation in which Δt is the time since the arrival of the shock at the node and d is the distance from the node to the origin

point:

$$P = \frac{scale \cdot peak(d) \cdot \Delta t}{rise(d)} \cdot \exp\left(1 - \frac{\Delta t}{rise(d)}\right) \tag{8.9}$$

8.3.6 Global Contact (3D global algorithm)

ALEGRA uses the ACME library [5] for the 3D contact algorithms (2D contact is not available). Contact is not currently supported with adaptivity or dynamic load balance. The "contact surface" for ACME is made up of a collection of sidesets, element block exteriors and analytic surfaces. The contact behavior between these entities can be specified pair by pair or at once through default data. The frictional behavior of the contact is specified using friction models (analogous to material models for elements). Data must be specified for every pair of sidesets, element blocks and analytic surfaces in the problem. Suitable defaults do not exist that will work for all problems so the user is required to input at least default data.

```
GLOBAL CONTACT                                              (3D only)
   [SIDESET | BLOCK] id
   :
   [additional SIDESET or BLOCK keywords]
   :
   [ANALYTIC SURFACE]
      TYPE = [PLANE | CYLINDER INSIDE | CYLINDER OUTSIDE | SPHERE]
      [Type Specific Subkeywords (see the following Tables)]
   [END]
   :
   [additional analytic surface keywords]
   :
   [ENFORCEMENT ITERATIONS int]
   [TRIVIAL MASS real]
   [DEFAULT DATA]
      [SEARCH NORMAL TOLERANCE real]
      [SEARCH TANGENTIAL TOLERANCE real]
      [KINEMATIC PARTITION [AUTO | real] ]
      [FRICTION MODEL ID int]
   [END]
   :
```

```
   [DATA [SIDESET|BLOCK|ANALYTIC SURFACE] int,
         [SIDESET|BLOCK|ANALYTIC SURFACE] int]
      [SEARCH NORMAL TOLERANCE real]
      [SEARCH TANGENTIAL TOLERANCE real]
      [KINEMATIC PARTITION [AUTO | real] ]
      [FRICTION MODEL ID int]
   [END]
   :
   [additional data blocks]
   :
   [FRICTION MODEL ID int]
      [FRICTIONLESS | CONSTANT | TIED | SPOT WELD |
         PRESSURE DEPENDENT | VELOCITY DEPENDENT]
      [Type Specific Keywords (see the following Tables)]
   [END]
   :
   [additional friction model keyword blocks]
   :
END
```

This keyword group identifies the surfaces that may come into contact with other similarly specified surfaces (or itself) during the calculation. Contact of a surface with itself is allowed and the surface must be specified if this is anticipated. By using this keyword block, the user ensures that the specified surfaces will not interpenetrate, but instead will exert a force to prevent penetration. For convenience, a block-id may be specified instead of a sideset so that all exterior surfaces of that block will be treated as a contact surface. However, if parts of the block will never come into contact with other surfaces, this can result in significant inefficiencies and slow down a calculation unnecessarily.

Table 25: Subkeywords for analytic surfaces.

Type	Sub-keywords	Argument	Description
`PLANE`	`POINT`	`X realY realZ real`	The x, y, z locations of a point in the plane.
	`NORMAL`	`X realY realZ real`	The x, y, z components of a vector normal to the plane.

continued on next page

continued from previous page			
CYLINDER {INSIDE | OUTSIDE}	CENTER	X realY realZ real	The x, y, z locations of the center of the cylinder.
	AXIS	X realY realZ real	The x, y, z components of a vector defining the axis of the cylinder.
	RADIUS	real	The radius of the cylinder.
	LENGTH	real	The length of the cylinder.
SPHERE	CENTER	X realY realZ real	The center of the sphere.
	RADIUS	real	The radius of the sphere.

Table 26: Subkeywords for friction models.

Type	Subkeywords	Argument	Description
FRICTIONLESS			There are no subkeywords.
CONSTANT	FRICTION COEFFICIENT	real	The coefficient of friction.
TIED			There are no subkeywords.
SPOT WELD	NORMAL CAPACITY	real	The normal force capacity of the spot weld.
	TANGENTIAL CAPACITY	real	The tangential force capacity of the spot weld.
	FAILURE STEPS	int	The number of time steps over which the failure will occur.
	FAILED MODEL ID	int	The ID of the friction model to be used once failure has occurred.
PRESSURE DEPENDENT	FRICTION COEFFICIENT	real	The coefficient of friction.
	REFERENCE PRESSURE	real	The reference pressure for the pressure dependence.
	PRESSURE EXPONENT	real	The exponent to be used in the pressure dependence.
VELOCITY DEPENDENT	STATIC COEFFICIENT	real	The coefficient of friction if the two surfaces are "sticking".
	DYNAMIC COEFFICIENT	real	The coefficient of friction if the two surfaces are "sliding".
	VELOCITY DECAY	real	The decay exponent to transition between the static and dynamic coefficients of friction.

An arbitrary number of "surfaces" may be specified along with a mixed set of sidesets, blocks, and analytic surfaces. For example, to specify the surfaces given by sidesets 11 and 12 plus the exterior surfaces of block 3 with frictionless contact, the minimum input would appear as the following:

```
global contact
   sideset 11
   sideset 12
   block 3
   default data
      friction model id = 1
   end
   friction model id 1 frictionless
      $ no subkeywords
   end
end
```

To specify parameters (*e.g.*, friction coefficients) that govern how a pair of surfaces will interact, the `DATA` (or `DEFAULT DATA`) keyword subgroup should be used. The `KINEMATIC PARTITION` factor determines the master/slave relationship. The default behavior is to use the automatic (`AUTO`), dynamic determination built into ACME. This can be overridden with a specific value but should be avoided unless the user is very familiar with the ACME algorithms. A poor choice of `KINEMATIC PARTITION` value by the user can cause energy generation, and in extreme cases element inversion. A value of 0 makes the first surface the complete master and the second the complete slave. The opposite is true for a value of 1.

The explicit transient dynamic enforcement provided by ACME includes an iterative capability to increase the accuracy of the contact forces. By default, the number of `ENFORCEMENT ITERATIONS` is set to 5. This has been found to be a good trade-off between speed and accuracy. The ACME enforcement has an implicit assumption of Lagrangian meshes. The implication is that every node MUST have a non-zero mass. For `ALE` problems, it is possible to have zero mass nodes (i.e., void filled elements). To account for this, ALEGRA assigns a `TRIVIAL MASS` to all zero mass nodes to prevent division by zero. The default (dimensional) value is 1.0e-12 but can be reset with the `TRIVIAL MASS` subkeyword.

8.3.7 Cavity Expansion

```
CAVITY EXPANSION, sideset
   RADIUS TYPE = [SPHERICAL | CYLINDRICAL]
   BODY AXIS, modeset , nodeset
   TIP RADIUS = real
   BEGIN LAYER
      TOP = real
      BOTTOM = real
      PRESSURE COEFFICIENTS, real, real, real
      SURFACE EFFECT MODEL ID, int
   END LAYER
   SURFACE EFFECT MODEL, int , [SIMPLE ON OFF]
      $ the following are input for the SIMPLE ON OFF model
      FREE SURFACE TOP = real
      FREE SURFACE BOTTOM = real
      TOP SURFACE COEFFICIENT = real
      BOTTOM SURFACE COEFFICIENT = real
   END
END
```

This model is described in a report, "CavityExpansion: A Library for Cavity Expansion Algorithms, Version 1.0" [6]. This document describes the model and the validation done for the model. The user interface to the model is described here. Cavity expansion is a method for modeling the penetration of an axisymmetric or wedge-shaped solid body - a penetrator - into a target by using analytic expressions to capture effects of the target on the body. The target material is not actually modeled.

There is an assumption that the user must setup the problem such that the normal to the target is in the +z direction. Multiple CavityExpansion Boundary Conditions are allowed in a problem. The user can specify as many layers in the target as desired and also as many surface effect models as desired. However, the only surface effect model that is currently implemented is the "`SIMPLE ON OFF`" model. The body axis nodesets are single node nodesets that are used to define the axis of the penetrator.

8.4 Dynamics Algorithm Control

8.4.1 Bulk (Pronto) Artificial Viscosity

```
PRONTO ARTIFICIAL VISCOSITY [int]
   [LINEAR real (0.15)]
   [QUADRATIC real (1.2)]
   [TENSION LINEAR={OFF|ON} (OFF)]  (unstructured option only)
   [TENSION QUADRATIC={OFF|ON} (OFF)] (unstructured option only)
END
```

Artificial viscosity is added to shock problems to "smear" the shock front across several elements and prevent numerical oscillation behind the shock. The optional integer field identifies an instance of the model that can be accessed selectively by different blocks in the problem (see the `BLOCK` keyword).

The `TENSION LINEAR` keyword causes the linear component of the artificial viscous pressure to be set when the material is in tension. The `TENSION QUADRATIC` keyword causes the quadratic component of the artificial viscous pressure to be set when the material is in tension (i.e to have a non-zero value in elements in tension).

It is generally not appropriate to leave the quadratic or linear artificial viscous pressure on in tension. However, it has been recommended [2] that the linear artificial viscosity be left on for elastic materials and turned off for ideal gases in tension. These options are only available in the unstructured code for testing purposes and are inactive in the structured code.

Tim Trucano has provided some further guidance on the use of linear viscosity in shock problems. Artificial viscosity was originally introduced specifically for the purposes of "shock capturing," in other words, for turning the mathematical discontinuities presented by shock waves as solutions to the Euler equations into finite width viscous waves that obeyed the same "jump conditions" and entropy production as the original discontinuities. Any use of artificial viscosity in a shock fitting code such as ALEGRA that is not directed at shock smearing is problematic at best. Shock waves exist in normal fluids only in compression. Thus, a dissipation intended to only smear shock waves makes no logical sense if applied in tension. If users choose the artificial viscosity to be on in tension they should beware of the results. There are certain materials which manifest so-called rarefaction shock waves. These

are true mathematical discontinuities (at least to the limit of experimental observation) that exist in tension. These very special waves are a result of phase transitions, such as solid-solid phase transitions in certain geologic materials.

One might suppose that to capture rarefaction shocks, an artificial viscosity might need to be deployed in tension. However, we make no claim relative to the mathematics and numerics of this idea for the purposes of this manual and consider that this topic remains open for further study.

8.4.2 Hydrodynamics Cell Doctor

```
HYDRO CELL DOCTOR
   [TIME STEP RATIO real]
   [SOUND SPEED RATIO real]
   [VOLUME FRACTION CUTOFF real]
   [all CELL DOCTOR options]
END
```

`HYDRO CELL DOCTOR` is a specialization of the `CELL DOCTOR` keyword (Section 5.11 on page 135) applicable to `HYDRODYNAMICS` and its children. In addition to the facilities provided by `CELL DOCTOR`, the `HYDRO CELL DOCTOR` keyword provides facilities with which the user can attempt to alleviate the problem of a small fragment with a completely unrealistic thermodynamic state controlling the time step of the entire problem. The idea is that a very small fragment with a completely unrealistic sound speed should be removed from the problem. Logic has been added to the code to allow the user to eliminate such small fragments from the calculation.

Note: HYDRO CELL DOCTOR is not available for structured mesh physics options.

One does not have to enter all the data above; however, it is unlikely that the default values will allow any material to be removed. Each of the time step, sound speed, and volume fraction conditions must be met to result in the removal of material. Once the fragment is removed, its volume fraction is proportioned out to the other materials in the cell, including void.

The `TIME STEP RATIO` subkeyword specifies the ratio of the two smallest cell time steps. For example, if cell A has a limiting time step of 1.0e-6 and

cell B has the next smallest time step of 1.0e-5 then the ratio would be 0.1. Therefore the value specified should be between 0 and 1, where the smaller the value, the more unlikely that a fragment will be removed.

The `SOUND SPEED RATIO` subkeyword specifies the ratio of the two largest material sound speeds in the limiting cell. The value specified should be greater than 1, where the larger the value, the more unlikely that a fragment will be removed.

The `VOLUME FRACTION CUTOFF` subkeyword specifies the maximum fragment size that can be removed. The value specified should be between 0 and 1, where the smaller the value, the more unlikely that a fragment will be removed.

From the user input above, it is obvious that a user could easily empty the time step limiting cell every time step and eventually totally destroy the simulation. Therefore, extreme care must be used when this option is employed in order to maintain a credible calculation.

8.4.3 Maximum Volume Change Time Step Control

`MAXIMUM VOLUME CHANGE real(0.2)`

This keyword specifies the maximum volume change permitted for an element in a single cycle, as a fraction of the original element volume. The time step will be limited to ensure that this volume change limit will be observed.

8.4.4 Minimum Element Side Time Step Control

`MINIMUM ELEMENT SIDE TIMESTEP CONTROL`

The internal calculation of the time step also may underestimate the maximum stable time step because of a conservative calculation of the characteristic element length. A typical use of the `MINIMUM ELEMENT SIDE TIMESTEP CONTROL` keyword would be in Eulerian calculations, where the analyst may designate that the minimum element side be used in the calculation of the maximum stable time step, which will typically result in time steps similar

to what the CTH algorithm would produce. For Lagrangian or ALE calculations, however, use of this option may not give stable behavior if the mesh becomes too distorted by shear.

The following example will cause the time step used in a calculation to be one half of the value ALEGRA would normally use and will use the minimum side length for the characteristic dimension of an element.

```
time step scale = 0.5
minimum element side time step control
```

8.4.5 Void Compression

```
VOID COMPRESSION, [ON|OFF]
```

In an element subject to compression and containing void and material, the material will not be subject to compression until the volume change produced by the compression equals or exceeds the volume of the void. When active, this algorithm avoids premature or nonphysical loading of material that is on the boundary of a body.

This option is `OFF` by default because there are cases where it can induce nonphysical behavior. In particular, cases where no compression exists, such as in the ballistics of jets of material (shaped charge jets), slight perturbations can induce the void compression algorithm to generate erroneous response that can lead to nonphysical behavior. Thus, in cases where a body is primarily stress-free, this algorithm should remain off.

This option should be turned `ON` in cases where significant compression of elements containing void are expected (such as contained detonations). Turning on this option prevents material in largely empty cells from experiencing nonphysical stress states that can lead to element inversion.

8.5 Dynamics Supplementary Models

8.5.1 Programmed Burn

```
PROGRAMMED BURN
```

```
   [NONBURNING ELEMENTS ALLOWED]
   MATERIAL int
      DETONATION {POINT, vector, |
                  LINE,  vector1, vector2, |
                  PLANE, vector1, vector2, vector3 }
                  AT TIME real, BURN RADIUS real
      [BURN TRANSITION RATIO real]
      [BURN FRONT THICKNESS real]
   :
   [additional materials and detonation models]
   :
END
```

The programmed burn option allows simulation of a detonation by considering the DETONATION {POINT | LINE | PLANE} to be an energy source. Detonation times are calculated from information provided with the detonation object to determine the detonation velocities throughout the burn material. If a burn due to pressure loading or shock is desired, then one of the KEOS SESAME two-state burn models is recommended (see Section 12.9 on page 270).

If some of the burn material is discontinuous from the rest, burn times will not be calculated and detonation would not be simulated in the discontinuous section. The default behavior is to stop the calculation from proceeding. The NONBURNING ELEMENTS ALLOWED keyword is an optional flag that allows the calculation to proceed if detonation times cannot be calculated for some of the PROGRAMMED BURN MATERIAL. If this keyword is used, it must be the first keyword in the PROGRAMMED BURN input section; it will apply to all PROGRAMMED BURN MATERIALs.

The MATERIAL number must be a valid material id that incorporates the the KEOS JWL burn model with BRN = 1 (this is the preferred method), or the PROGRAMMED BURN JWL model (soon to be deprecated). Multiple burn MATERIALs are allowed; the input for each MATERIAL is considered complete when the subsequent MATERIAL keyword is read, or when the END keyword is read.

At least one detonation object (POINT, LINE, or PLANE) must follow the MATERIAL input. The DETONATION {POINT | LINE | PLANE} command specifies the location of the primer for a high explosive, the time (AT TIME) of detonation, and the BURN RADIUS within which no obstacles or corners

will be circumvented. Within the BURN RADIUS, the radial distance between each cell and the detonation point is used to calculate the detonation time. (Thus, any intervening non-explosive materials are completely ignored within the BURN RADIUS). The BURN RADIUS must encompass at least one vertex of an element in the material for which the detonation object is defined. If no cells are found to be within the region defined by the detonation object plus BURN RADIUS, the default behavior is to force a fatal code error.

Detonation points may be specified for use in one, two or three dimensional geometries. DETONATION LINEs are specified by the two end points of the line. Lines may be specified for use in two or three dimensional geometries. DETONATION PLANEs are specified by any three points in the plane; the plane is assumed to be of infinite extent. DETONATION PLANEs may only be specified for use in three dimensional geometries.

The BURN TRANSITION RATIO (default=2.) is the ratio applied to the amount of energy added in a cycle (less than one decreases the time for burn, greater than 1 increases the time). The BURN FRONT THICKNESS (default=2.) is the factor used to multiply the largest element side. The result is applied to the amount of energy added in a cycle (less than 1 decreases the time for burn, greater than 1 increases the time). If the aspect ratio of the elements within the burn region is high, this parameter can be used to adjust the burn front to the width of two elements in the burn direction.

Note that ALEGRA provides for the detonation front to "turn a corner" so that the minimum route along cell centers is calculated within the burn material region, outside the BURN RADIUS. This is not the minimum radial distance but provides satisfactory burn contours for many problems of interest. If the burn material is not continuous, an additional detonation point with the appropriately modified detonation time will be required in each discontinuous region if detonation is to be simulated in that region. The detonation velocity and other burn material parameters are input with the PROGRAMMED BURN JWL model or the KEOS JWL input set.

Note that, if the explosive can be detonated by the shock wave alone (reactive burn model) and the shock propagates through the material surrounding the explosive at a speed comparable to the detonation front, the reactive burn configuration may be a better representation of the problem.

8.5.2 Inter-material Fracture

```
FRACTURE
   VOID, real
   MIXED, real
END
```

The inter-material fracture option allows material in multi-material elements to separate. Normally, in a multi-material element, materials behave as in a welded fashion. Thus, if two materials are moving apart from each other in an ALE or Eulerian mesh, tension will form in the elements containing the material interfaces and work to restrain the motion. By allowing `INTERMATERIAL FRACTURE`, tension at the material interfaces is limited by allowing the materials to fracture in that element, effectively allowing the materials to separate. The `VOID` and `MIXED` inputs specify the pressure response in mixed elements containing void and material or multiple materials, respectively.

This algorithm also looks at adjacent elements connected through a face. If the element is adjacent to an element of a different material, the tensile pressure limits are also applied. This allows for inter-material fracture between an ALE or Eulerian element and a single material Lagrangian element where the multi-material condition would never be met, but separation of different materials is allowed.

The values of the fracture pressures are negative, indicating tensile pressure. Also, fracture material models must be defined for all materials.

9 Solid Dynamics Input

```
SOLID DYNAMICS
   ...
   [solid dynamics keywords]
   ...
   [HYDRODYNAMICS]
      ...
      [hydrodynamics keywords]
      ...
   [END]
   [DYNAMICS]
      ...
      [dynamics keywords]
      ...
   [END]
   ...
END
```

SOLID DYNAMICS is one of the options described in mesh physics choice Sections 5.2 and 5.3. The HYDRODYNAMICS and DYNAMICS sub-keywords and their corresponding END keywords are optional and may be included in the input file for clarity. Otherwise, all HYDRODYNAMICS/DYNAMICS keywords from Section 8 on page 150 and following may be included directly into the SOLID DYNAMICS keyword block.

9.1 Solid Dynamics Algorithm Control

9.1.1 Kinematic Error Catching

```
IGNORE KINEMATIC ERRORS, [ RESET TO IDENTITY |
                           LIMIT SMALLEST EIGENVALUE [,real] ]
                         [,SILENT]
```

This input allows the code to continue after a negative eigenvalue in the stretch tensor is detected. This error is indicative of a breakdown in modeling the solid kinematics. This option is purely for user convenience in pushing

calculations to completion. Upon detection of such a tensor, or optionally a tensor which has an eigenvalue below a specified floor, the code implements the desired fix and generates a warning.

If `RESET TO IDENTITY` is specified, the code sets the stretch tensor to the unit tensor in each element whose stretch is not positive definite. This option is computationally cheap but may induce large errors. This is the default behavior. Specifying a floor will have no effect.

If `LIMIT SMALLEST EIGENVALUE` is specified, the code checks the eigenvalues of each stretch tensor and implements the given `real` floor, or 1.0e-6 if none is specified.

`SILENT` suppresses warnings. This option limits the amount of error messages sent to standard out and is quite important if the `LIMIT SMALLEST EIGENVALUE` option is chosen, since this may result in the element generating an error message each time it is computed.

9.1.2 Kinematic Update Method

```
VR UPDATE METHOD, { FIRST | SECOND } ORDER
```

This input allows the user to choose whether to use a first or second order accurate method for updating the stretch and rotation tensor. The second order method entails an extra tensor inversion and a couple of floating point operations per element per time step. By default the first order method is used, but preliminary data indicates that the cost/benefit ratio of the second order is good, so it may become the default method in the future.

10 Adaptivity Input

```
ADAPTIVITY SPECIFICATION
   ENABLE ADAPTIVITY
   JUMP METRIC
      [subkeywords]
   END
   ELEMENT BUDGET
      [subkeywords]
   END
   LAYERING
      [subkeywords]
   END
   BLOCK ADAPT LEVELS
      ...
   END
   UNREFINEMENT CONTROL
      [subkeywords]
   END
END
```

ALEGRA provides the capability of dynamically refining the mesh during a simulation. This refinement, which divides elements into uniformly smaller subunits, is called H-adaptivity. Currently, this feature is provided only for 2D quadrilateral meshes and 3D hexahedral meshes. In two dimensional calculations, a quadrilateral element will be divided into four smaller quadrilaterals and, in three dimensions, a hexahedron will be divided into 8 uniformly smaller hexahedral elements.

This capability can be specified to take place at simulation initiation, via the `INITIAL REFINEMENT` keyword in the `DOMAIN` specification (Section 5.10.3 on page 132).

Refinement can also take place dynamically as the simulation is run. In the latter case the adaptivity option is controlled by user selections in the `ADAPTIVITY SPECIFICATION` input section, which is placed outside of the overall physics input specification.

The dynamic refinement is driven by error metrics that are computed in each element in specified regions. Those elements exceeding some limit for

the error metric value will be refined, subject to limitations on the number of levels of refinement desired by the user. Similarly, refined elements where the error metric falls below a threshold value will be removed and combined into the element one level higher in the refinement hierarchy.

Adaptivity is not available for structured mesh physics options.

In this section of input, the controls on the dynamic refinement are specified. ALEGRA currently supports only one error metric to control mesh refinement. This is the traction jump error metric. The traction is defined at an element face and is the dot product of the element stress tensor and the outward normal to the face. The traction jump at a face is then the difference in traction computed using the stress tensors in the two elements sharing a face. The error metric for an element is the average face traction jump. (Note that specialized versions of ALEGRA have included other error metric controls specific to the additional physics being modeled by those special versions.)

The user of adaptivity should note that when adaptivity is being used, only `sidesets` should be used to specify constraints in the region of the mesh in which adaptivity may be changing element connectivity and adding elements. This is because ALEGRA currently does NOT support refined `nodesets` and thus a constraint specified on a `nodeset` will NOT be applied to the newly added nodes on that mesh boundary. If the boundary section is not in a region of mesh where adaptivity may be changing the mesh, `nodesets` can still be used to specify constraints (see Section 7.1 on page 144).

10.1 Adaptivity Algorithm Control

10.1.1 Enable Adaptivity

The `ENABLE ADAPTIVITY` keyword indicates to ALEGRA that the adaptivity features will be active for the simulation. The presence of this keyword in the `ADAPTIVITY SPECIFICATION` input section is a requirement for adaptivity to operate in a simulation.

10.1.2 Jump Metric

```
ADAPTIVITY SPECIFICATION
   JUMP METRIC
      [JUMP REFINE THRESHOLD real]
      [JUMP UNREFINE THRESHOLD real]
      [JUMP NORMALIZER real]
      [JUMP TRACTION SCALING]
      [JUMP TRACTION FLOOR real]
   END
END
```

The `JUMP METRIC` measures the traction difference across a surface of an element. The magnitude of this difference is maximized across all surfaces of the element to obtain a value on the element. The resulting value can be used to indicate error on the element. To use this value to drive the adaptivity, refinement and unrefinement thresholds must be defined. There are two ways to do this: set the `JUMP REFINE THRESHOLD` and `JUMP UNREFINE THRESHOLD` to traction values that the problem is known to exhibit (if given, the `JUMP NORMALIZER` will divide all values before applying the thresholds), or indicate `JUMP TRACTION SCALING` which will pick a normalizer based on the average value of the tractions across the whole mesh.

An example input is:

```
adaptivity specification
   jump metric
      jump refine threshold   0.75
      jump unrefine threshold 0.50
      jump traction scaling
      jump traction floor     10.0
   end
end
```

which would automatically scale the traction values to the average over the whole mesh and ignore values that are less than 10.0 (which can be used to prevent the adaptivity from refining on "noise"). If the magnitude of the pressures or the tractions is approximately known in the problem, one can use input like this:

Table 27: Adaptivity Keywords for `JUMP METRIC`.

Sub-Keyword	Input	Description
`JUMP REFINE THRESHOLD`	`real`	Value of traction jump error value above which to refine an element
`JUMP UNREFINE THRESHOLD`	`real`	Value of traction jump error value below which to unrefine a previously refined element
`JUMP NORMALIZER`	`real`	Value by which to divide the traction jump error values to obtain a value to compare to refine/unrefine thresholds
`JUMP TRACTION SCALING`		Flag keyword to indicate that the jump error values are to be scaled by the average value found among all faces
`JUMP TRACTION FLOOR`	`real`	Minimum value of the jump error metric above which comparisons to the refine/unrefine thresholds will occur.

```
adaptivity specification
   jump metric
      jump refine threshold   0.75
      jump unrefine threshold 0.50
      jump normalizer         1.e8
   end
end
```

which will divide all element traction values by 1.e8 before applying the thresholds.

10.1.3 Element Budget

```
ADAPTIVITY SPECIFICATION
   ELEMENT BUDGET
      [MAXIMUM ELEMENTS int]
      [MAXIMUM LOCAL ELEMENTS int]
```

```
      [NUMBER OF ERROR BINS int]
      [APPLY TO JUMP ERROR]
      [JUMP WEIGHT FACTOR real]
      [JUMP ERROR VALUE FLOOR real]
   END
END
```

The ELEMENT BUDGET method is not an error indicator but instead limits the number of elements that can exist during the simulation. If the element budget is reached, the elements with the higher error values are refined before the elements with a lower error value. At this point, the element budget control can be used in conjunction with the traction jump error indicator.

Table 28: Adaptivity Keywords for ELEMENT BUDGET.

Sub-Keyword	Input	Description
MAXIMUM ELEMENTS	int	The maximum number of elements to ever be used in the simulation
MAXIMUM LOCAL ELEMENTS	int	The maximum number of elements to be used on any one processor in a parallel simulation
NUMBER OF ERROR BINS	int	Level of granularity of the error value sorting algorithm (higher values modestly increase the amount of parallel communication)
APPLY TO JUMP ERROR		Flag keyword to indicate that the refinement prioritization will be based on the traction jump error
JUMP WEIGHT FACTOR	real	(Not applicable with only one error indicator)
JUMP ERROR VALUE FLOOR	real	A cutoff below which elements are never refined

An example input is:

```
adaptivity specification
   element budget
      maximum elements 10000
      apply to jump error
```

```
      jump error value floor 1.e2
   end
end
```

which indicates that there should not be more than 10,000 elements, apply it to the jump error indicator for prioritization, and even if there are less than 10,000 elements, do not refine elements if the jump error value is less than 100.

10.1.4 Adaptivity Layering

```
ADAPTIVITY SPECIFICATION
   LAYERING
      HYDRO JUMP WIDTH int
   END
END
```

`LAYERING` spreads the adaptation of elements out over a user selected width of elements, leading to a more gradual onset of higher resolution. The number input after the keyword `HYDRO JUMP WIDTH` is the number of elements over which to spread out the refinement.

10.1.5 Block Specific Control

```
ADAPTIVITY SPECIFICATION
  BLOCK ADAPT LEVELS
     int int, [int int,] ... [int int]
  END
END
```

The `BLOCK ADAPT LEVELS` keyword is followed by one or more pairs of integers. In each pair of integers, the first number refers to a mesh block while the second refers to the maximum adaptivity level that will be allowed in the block. This is a method of overriding the maximum levels of adaptivity that is set for the whole mesh in the `DOMAIN` keyword section.

10.1.6 Unrefinement Control

```
ADAPTIVITY SPECIFICATION
   UNREFINEMENT CONTROL
      [CYCLE LAG int]
      [TIME LAG real]
      [LOCK INITIAL REFINEMENT,
         [UNTIL CYCLE int | UNTIL TIME real | NEVER UNREFINE] ]
      [REDUCE MAX REFINEMENT]
         [AT TIME real | AT CYCLE int]
   END
END
```

The adaptivity options in the `UNREFINEMENT CONTROL` input allow the user to control when unrefinement takes place. The first two controls are `CYCLE LAG` and `TIME LAG`. These two values control a delay between the indication of unrefinement by the error metric and the actual unrefining of the element. `CYCLE LAG` introduces a specific number of computational cycles between the unrefine indication and the actual unrefinement of an element, while `TIME LAG` introduces a delay of a set amount of problem time between the indication and the action. Both features are present to reduce noise in the unrefinement of elements.

The `LOCK INITIAL REFINEMENT` flag is tied to the next three options: `UNTIL CYCLE`, `UNTIL TIME` and `NEVER UNREFINE`. These options allow the user to control the unrefinement of the refined mesh introduced during an `INITIAL REFINEMENT` step. (`INITIAL REFINEMENT` is controlled by the `DOMAIN` section of the ALEGRA input (Section 5.10.3 on page 132). The `LOCK INITIAL REFINEMENT` input forces the initially refined mesh to remain at its level of refinement until a certain computational cycle is reached (`UNTIL CYCLE`) or until a given problem time is reached (`UNTIL TIME`). The `NEVER UNREFINE` keyword forces the initial refinement to persist indefinitely.

The `REDUCE MAX REFINEMENT` keyword provides the user with the capability to manually force an unrefinement of one level everywhere in the mesh at a particular time or cycle. When used with the block-specific `INITIAL REFINEMENT` keyword driven capability in the `DOMAIN` section (see Section 5.10.3 on page 132), the user can force the problem mesh to logically match the mesh in a target region that is used in the staged activation of regions process (see Section 5.4.2 on page 96).

Table 29: Adaptivity Keywords for `UNREFINEMENT CONTROL`.

Sub-Keyword	Input	Description
`CYCLE LAG`	`int`	number of cycles between an unrefinement indication from the error metric and the actual element unrefinement
`TIME LAG`	`real`	amount of problem time between an unrefinement indication from the error metric and the actual element unrefinement
`LOCK INITIAL REFINEMENT,`		Flag keyword
`UNTIL CYCLE`	`int`	computational cycle at which an initially refined mesh can unrefine
`UNTIL TIME`	`real`	computational time at which an initially refined mesh can unrefine
`NEVER UNREFINE`		Flag keyword
`REDUCE MAX REFINEMENT,`		Flag keyword
`AT CYCLE`	`int`	computational cycle at which an initially refined mesh will unrefine by one level everywhere
`AT TIME`	`real`	computational time at which an initially refined mesh will unrefine by one level everywhere

10.2 Dynamic Load Balancing

To keep the load balanced over many processors during adaptivity, a load balancer can be invoked. This will produce a mesh restructuring. In the current release, load balancing is either on or off and is called for each time step although it may decide that no mesh movement is required. Load balancing is turned on with the following syntax:

```
HYDRODYNAMICS
   ...
   LOAD BALANCE
   END
END
```

That is, it resides in the physics specification block.

Note: LOAD BALANCE is not available for structured mesh physics options.

11 Structured Mesh Input

11.1 Structured Mesh

```
STRUCTURED MESH
   [subkeyword-list]
END
```

The STRUCTURED MESH block is one method of specifying a computational domain for a MBS problem.

The STRUCTURED MESH specification consists of one of the following keyword -- end pairs and the associated keywords and values. The AMR -- END keyword pair permits the specification of simple Cartesian mesh that is generated inline. The AMR keyword requires one argument that specifies the geometry as 2D, 3D and the geometry as either cylindrical or rectangular. The SET ASSIGN -- END keyword pair allows specification of nodesets and sidesets on the exterior of a STRUCTURED MESH specified domain.

An example of the STRUCTURED MESH -- END syntax is:

```
structured mesh
   amr 3dr
      periodic, x
      nx = 10
      ny = 10
      nz = 10
      bx = 40
      by = 7
      bz = 5
      gmin =  1.0 1.0 1.0
      gmax = 40.0 7.0 5.0
   end
   set assign
      nodeset,ihi,2
      nodeset,jhi,1
   end
end
```

11.1.1 AMR

```
AMR {2DR | 2DC | 3DR}
   [subkeyword-list]
END
```

The `AMR -- END` block pair surrounds the description of the geometry of an inline specified rectilinear mesh. The extent of the domain is given by a pair of vectors. The number of blocks in each coordinate direction and the number of elements in each block are given by additional keywords. The total number of elements specified in an inline specified mesh is the product of the total number of blocks `BX x BY x BZ` and the total number of elements per block `NX x NY x NZ`. For a 2D mesh `nz`, and `bz` must be omitted.

2DR - 2D Rectilinear

2DC - 2D Cylindrical

3DR - 3D Rectilinear

11.1.2 SET ASSIGN

```
SET ASSIGN
   [{NODESET | SIDESET} ,{ IHI | JHI | KHI | ILO | JLO | KLO }, int]
END
```

The `SET ASSIGN -- END` keyword pair allows the specification of simple nodesets and sidesets on the exterior of inline meshes.

11.2 Block

```
BLOCK start TO end
 [subkeyword-list]
END
```

The `BLOCK` keyword group allows the user to specify the materials that are contained in a block and the type of mesh movement desired in the

Table 30: Keywords for AMR -- END.

Sub-Keyword	Input	Description
DEBUG		Activates the output of debugging messages associated with the AMR grid.
PERIODIC,{X | Y | Z}	int	Specification of a periodic domain.
NX	int	Number of cells in x-direction.
NY	int	Number of cells in y-direction.
NZ	int	Number of cells in z-direction.
BX	int	Number of blocks for the unrefined grid in x-direction.
BY	int	Number of blocks for the unrefined grid in y-direction.
BZ	int	Number of blocks for the unrefined grid in z-direction.
GMIN	vector	Minimum coordinates (x,y,z) for the domain.
GMAX	vector	Maximum coordinates (x,y,z) for the domain.

block. The default is for a block to be a voided Lagrangian block (i.e., if all subkeywords are omitted). The BLOCK subkeywords are identical to the UNSTRUCTURED REGION BLOCK subkeywords.

For multi-block structured regions, the "start TO end" syntax identifies which subset of blocks for which the user is specifying block attributes. The "start TO end" BLOCK syntax may be used in conjunction with the "standard" block syntax to override the attributes of individual blocks.

11.3 Output

```
PLOT, {OUTPUT FORMAT}
 {FILE = ''file_name''}
END
```

There is no default mesh output produced from a MBS physics option problem. There are currently three OUTPUT FORMAT options: EXODUS, EN-

SIGHT ASCII, and ENSIGHT BINARY. The `FILE =` directive specifies the entire output file name for serial exodus output and provides a root name for parallel GENESIS and all ENSIGHT ASCII and ENSIGHT BINARY output.

The exodus files produced by parallel execution of structured mesh physics in ALEGRA do not contain the meta-data required for concatenation of the results into a single cohesive mesh. Alternatives for viewing the resulting EXODUS files include: importing multiple files into ENSIGHT, running ENSIGHT in "server of servers" mode over multiple files, and using a utility called EXJOIN that produces a combined file without equivalencing nodes. The EXJOIN utility is available from ALEGRA developers.

ENSIGHT output produced from a MBS run are in the ENSIGHT GOLD case file format. From parallel runs ALEGRA produces a file for each variable from each process as well as a geometry file from each processor and a single `case` file. These variable and geometry files from the individual processors can be joined together using a utility called ENSIGHTJOIN. This operation is very efficient because it is essentially the organized concatenation of the component files plus the appending of an index to each file. The meaning of ENSIGHT BINARY and ENSIGHT ASCII is self explanatory.

11.4 Mesh Input

```
MESH, {INPUT FORMAT}
 {FILE = ‘‘file_name’’}
END
```

There are two alternatives to inline mesh specification available to users of MBS physics. They are GENESIS and PLOT3D. The GENESIS input option is a degenerate form of the PLOT3D option since any GENESIS file appropriate for import into ALEGRA is suitable for translation into a PLOT3D file. When a GENESIS file is provided ALEGRA calls a library version of the GENTOP3D translator. This translation is carried out in serial and may require more memory resources than are available. For this reason large meshes should be translated into PLOT3D fies. It should be viewed as a convenience and may not be appropriate for large problems.

The PLOT3D file format is an extension of the PLOT3D geometry file format commonly used in other applications. As created by the GENTOP3D

utility for use in MBS ALEGRA it has appended to it additional information describing: the original block id's, the sideset and nodeset information, and block to block connection information. These enhanced PLOT3D files remain readable by ENSIGHT using the PLOT3D option.

The GENTOP3D translator is distributed along with the third party libraries required for compiling and linking alegra.

Two and three-D GENESIS files that satisfy the following restriction are suitable for translation into PLOT3D files for import into MBS ALEGRA: Each block of mesh in the GENESIS file must consist of a collection of elements that topologically resemble an array. In two dimensions this is a checkerboard pattern. In three dimensions a Rubik's Cube patter. The nodes between the blocks must be equivalenced for the solution to combine the blocks appropriately.

There are no restrictions on how these blocks may be connected to each other.

12 Material and Material Model Input

Specification of the materials and material models used for an ALEGRA calculation is expressed in rather general terms. A material, for purposes of an ALEGRA calculation, is defined by a set of material models and initial values of critical independent variables that may differ from the default reference values. Thus, ALEGRA input for material specification consists of a `MATERIAL` keyword group and corresponding sets of material `MODEL` subkeyword groups. The remainder of this section provides an overview of the material and material model input via examples. More detailed descriptions and syntax follow in the subsequent subsections. The final subsection contains complete examples of material and model input for all the material models.

As a simple example, the material and material model input that might be used to describe air for hydrodynamics calculations is shown below. For this simple material, only one material model is needed to describe the material response of interest. The initial density in this example was desired to be lower than the default, which would be equal to "rho ref".

```
material 200 air
   model 201
   density = 0.001  $ g/cm^3, initial density less than rho ref
end

model 201 ideal gas
   rho ref     = 0.0011756624 $ g/cm^3
   temperature = 298.         $ K
   gamma       = 1.4
   cv          = 0.7165e+07   $ erg/g/K
end
```

Generally, the material and material model identification numbers are arbitrary and for the convenience of the user. However, the model number specified in the `MATERIAL` keyword group must correspond to the model id of the material `MODEL` keyword group.

The string following the `MATERIAL` keyword and id is optional and is for the convenience of the user. All text up to a comment or end-of-line will be read as the optional string.

The optional string must not match any main or reserved keywords and must not be an integer or real value. It will be parsed as such and an error will result. To avoid this, the string may be placed within double quotes (see the string following material 200 in the example below, "6061" would be parsed as an integer if not quoted).

The comments denoting units which follow a number of the parameter lines are for the convenience of the user.

The initial values of independent variables (e.g., temperature and density) do not need to be specified unless the initial state of the material and the reference state of the model are different.

A more complex example of material and material model input for a solid dynamics calculation is shown below. The units in square brackets placed after the density are simply a demonstration of the code's capability of handling unit conversions of the input (see Section 4.1.3 on page 69).

```
material 100 tungsten penetrator
   model 101              $ elastic plastic
   model 102              $ mie-gruniesen us-up
   model 103              $ pressure-dependent fracture
   density 1850. [kg/m^3] $ demonstration of unit, not required
end

model 101 elastic plastic
   youngs modulus    3.42e12 $ dyne/cm^2
   poissons ratio    0.29
   yield stress      1.5e+10 $ dyne/cm^2
   hardening modulus 0.0     $ dyne/cm^2
   beta              1.0
end

model 102 mg us up
   c0      =   3.998e5 $ cm/s
   sl      =   1.237
   gamma0  =   1.54
   rho ref =  18.5     $ g/cm^3
   cv      =   2.5e10  $ erg/g/K
   pref    =   0.0
   tref    = 298.0
```

```
   e shift =   1.0e+10
end

model 103 frac presdep
   init frac pres     = -5.e10
   density tolerance  =  1.e-6
   pressure tolerance =  1.e+2
end

material 200 "6061-t6 aluminum target plate"
   model 201          $ elastic plastic
   model 202          $ keos mie-gruniesen
   model 203          $ pressure-dependent fracture
   density     = 2.66 $ not needed if same as r0, g/cm^3
   temperature = 298. $ not needed if same as t0, K
end

model 201 elastic plastic
   youngs modulus    = 73.00e+10 $ dyne/cm^2
   poissons ratio    = 0.3225
   yield stress      = 2.76e+09  $ dyne/cm^2
   hardening modulus = 1.24e+09  $ dyne/cm^2
   beta              = 1.
end

model 202 keos miegrun $ Newer version of mg us up
   cs = 5.328e5  $ cm/s
   sl = 1.338
   g0 = 2.18
   r0 = 2.66     $ g/cm^3
   cv = 1.034e07 $ erg/g/K
   t0 = 298.0    $ K
end

model 203 frac presdep
   init frac pres     = -1.e9  $ dyne/cm^2
   density tolerance  =  1.e-6
   pressure tolerance =  1.e+2
end
```

Note that each material is described by a set of material models. Generally, any set of material models may be specified for a material as appropriate for the physics performed in the ALEGRA calculation (see Sections 5.2 and 5.3 on pages 93 and 94). This generality provides substantial flexibility for the user to describe very complex material behavior. However, it also is relatively easy to specify inconsistent material models which produce invalid results. Limited consistency checking exists, but the user is encouraged to refer to the subsequent sections on the `MATERIAL` and `MODEL` keywords for more information on constructing consistent input.

All materials described by a `MATERIAL` keyword block may occupy any element in the mesh. Thus, all elements provide for the possibility of all materials being present. This capability is, in general terms, achieved by providing an array of `MATERIAL DATA` objects, one for each material defined in the user input file. These `MATERIAL DATA` objects contain the variables for each material in that particular element. The material models defined for the material operate on this data. Thus, knowledge of the variables operated on by the models comprising a particular material can provide useful insight into the nature and type of response of the models. These variables are listed in the tables of registered plot variables for each material model discussed in the following sections (see Table 31 in Section 12.2 on page 207).

In general, the element maintains an array of material volume fractions of all materials present within that element. Thus, the sum of all of the material volume fractions for an element must equal one. Furthermore, individual values of material volume fraction range from zero to one. In some cases an element has material volume fractions which sum to a value less than one. In such a case, a portion of the element is considered empty. A fictitious material called Void occupies the remaining empty volume of the element. Void is a material with no material models defined. Thus, it generally has no properties and variables. Void is essentially vacuum.

The users should be aware that there are a number of ways in which a material is deposited in an element (see Section 5.9.1 on page 116). At initialization, material may be assigned to all elements of an element block through the `MATERIAL` keyword in the `BLOCK` input. Note that multiple material keywords in the `BLOCK` input produce equal material fractions for each material. Alternatively, the `DIATOM` input also allows material to be inserted into elements meeting a spatial criteria (see Section 5.7.1 on page 100). During a calculation, material may enter or exit an element due to advection. In all of these cases, multi-material elements may occur, and void may also be

present.

12.1 Materials

```
MATERIAL int [string]
   MODEL int
   [MODEL int]
   ...
   [variable_name real]
   [variable_name real]
   ...
END
```

The `MATERIAL` keyword begins a block of user input specifying the models and initial state of a material to be used in the calculation. The `MATERIAL` keyword is followed by an identifying integer which refers to the particular material throughout the input specification file. For example, the `MATERIAL` which fills an element `BLOCK` must refer to a valid material id.

```
physics specification
   block 1
      material 201 $ see BLOCK input for this item
   end
end

material 201 "6061-T6 Aluminum"
   ...
end
```

In this example, the `BLOCK` input specification that `material 201` fills the element `block 1` corresponds to the `MATERIAL` keyword with `int` id `201`.

The identifying material number is followed by an optional `string` which may be used to describe the material. This `string` is parsed to the end of line or next valid keyword. Supplying the optional descriptive `string` improves readability of the input file. The `string` is printed to the output file of the calculation, also improving its readability, as well as used in various

error messages. In the above example, the descriptive `string` is `6061-T6 Aluminum`.

The `string` may or may not be enclosed in quotes. Recognition of the `string` as a material description assumes the string does not match any valid material keyword. For example, the string `Be` (short for beryllium) matches the material keyword `BETA` and an input error occurs. The ambiguity is removed by expanding `Be` to `Beryllium` or by surrounding "`Be`" with quotes.

12.1.1 `MODEL` subkeyword

```
MODEL int
```

In ALEGRA, a material model is a module which computes the values of a set of dependent material variables given a set of independent material variables and other appropriate data. For complex materials, a number of models may need to be sequenced for a material so that all variables of interest are computed. This can be very customized and useful for prototyping, carefully specifying parameters to capture the behavior of interest. Because it can also be error prone, in some cases a collection of models eventually evolved into a single model with a simpler user interface.

The `MODEL` subkeyword is used to specify a single model or construct a list of material models which will be used to describe the response of the material. Any number of models, as appropriate for the material and physics of the problem, may be specified. It is important to note that *THE ORDER OF THE MODELS SPECIFIED FOR A MATERIAL IS THE ORDER IN WHICH THE MODELS WILL BE APPLIED* in the calculation. For example, the input fragment:

```
material 10 Administratium
   model 11
   model 25
   model  8
end
```

applies models 11, 25, and 8, in that order, when computing the state of material 10. This ordering has implications for properly computing the state

of the material. The variables of some models may depend on variables computed in other models. This implies that models with such dependent variables be specified *AFTER* the models which compute those variables.

For example, if `MODEL 25` requires temperature as a known input value, `MODEL 11` must calculate temperature. Otherwise, the temperature used in `MODEL 25` will be the temperature from the preceding cycle (to make matters more complicated, in some cases this may be exactly what the model requires!).

To address the difficulty in specifying the correct order of material models in a material, all material model implementations specify all variables, identifying each as either input, input/output, or output. Thus, to properly use each model, all input variables of the model are provided to the material (via element data) or computed by a preceding material model. The tables in the material model subsections list all the variables and the corresponding input/output type.

12.1.2 `variable_name` subkeyword

```
variable_name real
```

The `variable_name` input specification is used to define a non-zero initial value for an independent variable of the material. The most common (and currently the only use of this facility) is to specify initial density and temperature of a material for equation of state material models. Currently, it is possible to initialize any scalar variable which can be written to the plotting database.

This facility is provided as a general capability for models which require non-zero initial values for certain scalar variables. The generality should make this capability usable for the most complicated material models. Using this facility is highly dependent on the particular material model and generally requires detailed knowledge of the implementation of the model. Thus, it is recommended that this capability not be used indiscriminately.

12.2 Material Models

```
MODEL int model_name
   parameter [{int | real}]
   parameter [{int | real}]
   ...
END
```

The `MODEL` keyword begins a block of input which specifies a particular material model and provides for input of the appropriate material model `parameters`. The `MODEL` keyword is followed by an integer identifier selected by the user, followed by the name of the material model to be used. The integer id is used to cross-reference the material model with the material model list contained in a `MATERIAL` keyword group.

The body of the material `MODEL` keyword group consists of `parameter` value input. The `parameters` are set by the name of the `parameter` followed by the `int` or `real` value, as appropriate. The `parameter` names and input value types are listed in the material model catalog in the subsequent subsections. The value following a `parameter` keyword is parsed as either `int` or `real`. Thus, an `int` cannot be coerced to a `real` value (e.g., 298 is an `int` and 298. is a `real`). Instead, a parsing error with an appropriate diagnostic will occur.

Examples:

```
material 1 elastic copper
   model 2
   model 1
   temper = 500. $ K, can change initial temp from ref value
end

material 2 elastic plastic copper
   model 3
   model 1
end

model 1 mg us up
   ...
end
```

```
model 2 linear elastic
   ...
end

model 3 elastic plastic
   ...
end
```

The `MODEL` name must be a valid name for an available model. Note that a particular `MODEL` may be used in more than one `MATERIAL` specification. Furthermore, models may also be included in the user input that are not used by any material. Such models are not used (although the input is parsed). This may be convenient when running problems from the same input deck but desiring to vary material models used by certain materials.

A list of the currently available models is provided in Table 31. The models are approximately categorized into general model types although in some cases, a model may contain more or less capability than is typically assumed for such models.

The following subsections provide individual descriptions of the material models available in ALEGRA. The models are described in the order in which they appear in Table 31. Each model will have two tables, one listing model input parameters and another listing the registered model variables.

In the model input parameter tables, the full parameter name for parsing is listed, followed by the parameter type, either `real` for floating point number or `int` for integer. A description of the parameter is also provided. Note that the initial value of any unspecified parameter defaults to zero, unless indicated otherwise in the parameter description. In the registered model variable tables, the variable name is followed by the variable type (`int`, `real`, `vector`, `symtensor`, `tensor`, or `antitensor`), variable mode (`INPUT`, `IOPUT`, or `OUTPUT`), and a description. In general, all variables in the model variable tables may be plotted by listing the variable name (without underscores) within the `PLOT VARIABLES` keyword construct (see Section 4.2.12 on page 81). Extra variables for each model that may be plotted by the HISPLT program are listed in an additional table where applicable. HISPLT variables that are not model-dependent for the various ALEGRA physics options are listed in Section 13.3 on page 298.

The variable mode `INPUT`, indicates that a particular variable is used as input to the model, implying that the value is current when the model is applied. The `IOPUT` mode indicates that a variable contains a current value upon entry to the model and is updated in the course of the model. The `OUTPUT` mode indicates that the variable is computed by the model. Generally, a consistent set and sequence of material models for a material will have all `INPUT` variables provided by the element or computed as `IOPUT` or `OUTPUT` variables in preceding models.

The directory and files containing source code for the model are also listed, relative to the `$ALEGRA_DEVEL` path. Many of the models are specialized and are only applicable to particular versions of the code or particular physics selections. Where possible, these requirements are noted, although this information may be incomplete.

Table 31: Material Model Types and Model Names.

General Material Model Type	Model Name
Equation of State	GENERIC EOS IDEAL GAS JWL KEOS Ideal Gas KEOS JWL KEOS MieGruneisen KEOS Sesame MG POWER MG US UP
Constitutive	ELASTIC PLASTIC LINEAR ELASTIC SOIL CRUSHABLE FOAM ISOTROPIC GEOMATERIAL
Yield	STEINBERG GUINAN LUND JOHNSON COOK EP ZERILLI ARMSTRONG BAMMANN CHIESA JOHNSON VON MISES YIELD
Plasticity	SIMPLE RADIAL RETURN EP RADIAL RETURN
Combined Models	CTH ELASTIC PLASTIC BFK CONCRETE
Fracture	FRAC PRESDEP
Reactive Burn	KEOS Arb KEOS Ffrb KEOS Hvrb KEOS Igrb KEOS Ptran
Burn	PROGRAMMED BURN JWL

12.3 Equation of State Models

Equation of state models relate the internal energy to other state variables (such as density and pressure). The equation of state models in ALEGRA range from simple models that exist only to provide required thermodynamic quantities (such as the `GENERIC EOS`) to complex models that extend the basic concepts of equilibrium thermodynamics to include the effects of microstructure and time-dependent behavior (such as the reactive burn models).

To simplify input for several of the equation of state models, a specially formatted ASCII file named `EOS_data` is distributed with ALEGRA that provides input parameters for many predefined materials. Incorporation of any particular data set into the `EOS_data` file does not imply correctness or validation of the data set, but it is intended as a starting point for materials of interest in the absence of experimental data for the particular application. Parameters in the `EOS_data` file are given in cgseV units, which are standard CTH units (the `EOS_data` file is also distributed with CTH). The data are read and automatically converted to problem units during ALEGRA's set up phase. The default location of the `EOS_data` file is set by the environment variable `$ALEGRA_MIGDATA`. All models in the `EOS_data` file are not necessarily accessible by the ALEGRA code, and conversely, all models in the ALEGRA code do not have data provided in the `EOS_data` file. For applicability to a specific constitutive model, look in the input data below for the `MATLABEL` input.

Models that are shared with CTH use the MIG (Material Interface Guidelines) [4]. These models have an associated data file that lists the required Fortran routines and other information specified in the guidelines. If this file is available with the ALEGRA source code, it is listed in these sections with the description of the material model. The actual source code that is shared with CTH is in an external library that is distributed with the ALEGRA code. Many of the equation of state models that are shared by CTH and ALEGRA are named with the prefix "`KEOS`." These are the recommended models for MieGruneisen and JWL although older versions of many of these models remain for backward compatibility.

Several of the equation of state models include an optional parameter called `ESHIFT`. This parameter is used to temporarily shift the zero energy condition or reference energy, `Eref`, up to a range where a solution algorithm is not likely to return negative energies and temperatures, which are

problematic for some codes. The KEOS models calculate a reasonable value of ESHIFT internally as an aid to the user, although it can be overridden by user input.

12.3.1 Generic EOS

```
MODEL model_id_number GENERIC EOS
   [parameter = value]
   ...
END
```

The generic equation of state is used for materials which have a mechanical stress tensor model but would otherwise have no computation of temperature or sound speed. Note that sound speed is required of all materials in the DYNAMICS hierarchy (*e.g.*, HYDRODYNAMICS, SOLID DYNAMICS). Thus, in the absence of any other model that computes sound speed for the material, this model should be specified. The specific heat, C_v, is taken to be a constant. A reference sound speed, C_{s0}, must be specified. This may be derived from elastic moduli specified in other models describing the material. Pertinent equations are shown below.

$$T = T_{ref} + \frac{E - E_{ref}}{C_v} \tag{12.10}$$

$$C_{s0} = \sqrt{\frac{K_0 + \frac{4}{3}G_0}{\rho_0}} \tag{12.11}$$

$$C_s = \sqrt{\frac{\rho_0 + C_{s0}^2}{\rho}} \tag{12.12}$$

- Modules: material_libs/standard_models
  - matmod_generic_eos.h
  - matmod_generic_eos.C
- Physics: solid dynamics

Table 32: Input Parameters for GENERIC EOS.

Parameter Name	Type	Description
RHO REF	real	Density at the reference state, ρ_0
TREF	real	Absolute temperature at the reference state, T_{ref}
CV	real	Specific heat at constant volume, C_v
EREF	real	Specific internal energy at the reference state, E_{ref}
REF SOUND SPEED	real	Sound speed at the reference state, C_{s0}
E SHIFT	real	Arbitrary shift of reference energy (optional)

Table 33: Registered Plot Variables of GENERIC EOS.

Variable Name	Type	Mode	Description
DENSITY	real	INPUT	Material density, ρ
ENERGY	real	INPUT	Specific internal energy, E
TEMPERATURE	real	OUTPUT	Absolute temperature, T
SOUND SPEED	real	OUTPUT	Sound speed, C_s

```
$ Sample input the generic eos model
$ A material with this eos would probably also specify some
$ strength model, such as elastic plastic.

model 32 generic eos            $ cgsK units
  rho ref         = 8.932       $ gm/cm^3
  tref            = 298.        $ K
  cv              = 3.924e+06   $ erg/gm/K
  ref sound speed = 4.4468e+05  $ cm/s
end
```

12.3.2 Ideal Gas

```
MODEL model_id_number IDEAL GAS
   [parameter = value]
   ...
END
```

The ideal gas model [21] computes the pressure, P, the temperature, T, and the sound speed, C_s, for the material using the standard ideal gas equations.

$$P = \rho(\gamma - 1)(E - E_{ref}) \tag{12.13}$$

$$T = \frac{E - E_{ref}}{C_v(1 + \bar{Z})} \tag{12.14}$$

$$C_s = \sqrt{\frac{\gamma P}{\rho}} \tag{12.15}$$

For plasmas, there can be electron contributions to the gas energy and pressure. To account for any electron effects, the `FIXED ZBAR` parameter is provided. Set `FIXED ZBAR` = 1.0 for single ionization, `FIXED ZBAR` = 2.0 for double ionization, etc. This parameter defaults to zero.

- Modules: material_libs/standard_models
  - matmod_idlgas.h
  - matmod_idlgas.C
- Physics: hydrodynamics

Table 34: Input Parameters for `IDEAL GAS`.

Parameter Name	Type	Description
`RHO REF`	`real`	Density at the reference state, ρ_0
`TREF`	`real`	Absolute temperature at the reference state, T_{ref}
`CV`	`real`	Specific heat at constant volume, C_v
`GAMMA`	`real`	Ratio of specific heats, $\gamma = C_p/C_v$
`FIXED ZBAR`	`real`	Fixed ionization state (to account for electrons)
`E Shift`	`real`	Arbitrary shift of reference energy (optional)

Table 35: Registered Plot Variables of IDEAL GAS.

Variable Name	Type	Mode	Description
SPECIFIC HEAT VOL	real	INITIALIZE	Initialized from C_v parameter
DENSITY	real	INPUT	Material density
ENERGY	real	INPUT	Specific internal energy
PRESSURE	real	OUTPUT	Pressure
TEMPERATURE	real	OUTPUT	Absolute temperature
SOUND SPEED	real	OUTPUT	Sound speed

```
$ Sample input for ideal gas model
model 11 ideal gas       $ cgsK units
  gamma   = 1.667
  rho ref = 1.624e-4     $ g/cm^3
  tref    = 300.         $ K
  cv      = 3.116324e+07 $ erg/g/K
end
```

12.3.3 JWL

```
MODEL model_id_number JWL
   [parameter = value]
   ...
END
```

This is the Jones-Wilkins-Lee equation of state for high explosives. The equations are described fully by Reference [21]. This model describes the equation of state of high explosive materials in a fully detonated state. The parameters are unique for a given undetonated density and temperature.

Input parameters must be in consistent (CGS or SI) units, which is not usually the way JWL parameters are published [10]. The JWL constants are often cited in the following system of units: A, C and PCJ are given in Mbar = 10^{12} GPa, density in gm/cm^3, DCJ in cm/μsec, and E0 in Mbar-cm^3/cm^3.

- Modules: material_libs/standard_models

  - matmod_jwl.h
  - matmod_jwl.C

- Physics: hydrodynamics

```
$ Sample input for JWL model
model 21 jwl           $ cgsK units
   a       = 4.63e+12  $ dyne/cm^2
   b       = 8.873e+10 $ dyne/cm^2
   c       = 1.349e+10 $ dyne/cm^2
   omega   = 0.35
   r1      = 4.55
   r2      = 1.35
   rho ref = 1.65      $ g/cm^3
   e shift = 1.e+10    $ erg/gm
   pcj     = 2.15E+11  $ dyne/cm^2
   dcj     = 7.03e+05  $ cm/s
   tcj     = 4062.0    $ K
   tref    = 298.0     $ K
end
```

Table 36: Input Parameters for `JWL`.

Parameter Name	Type	Description
`RHO REF`	`real`	Density in unreacted reference state
`TREF`	`real`	Temperature in unreacted reference state
`E SHIFT`	`real`	Arbitrary shift of reference energy (optional)
`A`	`real`	JWL parameter in units of problem
`B`	`real`	JWL parameter in units of problem
`C`	`real`	JWL parameter in units of problem
`OMEGA`	`real`	dimensionless JWL parameter
`R1`	`real`	dimensionless JWL parameter
`R2`	`real`	dimensionless JWL parameter
`E0`	`real`	JWL parameter in units of problem
`PCJ`	`real`	Chapman-Jouget pressure
`DCJ`	`real`	Chapman-Jouget detonation front velocity
`TCJ`	`real`	Chapman-Jouget temperature

Table 37: Registered Plot Variables of JWL.

Variable Name	Type	Mode	Description
SPECIFIC HEAT VOL	real	INITIALIZE	Initialized from C_v parameter
DENSITY	real	INPUT	Material density
ENERGY	real	INPUT	Specific internal energy per unit mass
PRESSURE	real	OUTPUT	Pressure
TEMPERATURE	real	OUTPUT	Absolute temperature
SOUND SPEED	real	OUTPUT	Bulk sound speed
DPDRHO	real	OUTPUT	Derivative of pressure with respect to density

12.3.4 KEOS Ideal Gas

```
MODEL model_id_number KEOS IDEAL GAS
   [parameter = value]
   ...
END
```

The Kerley EOS model for ideal gas [22] computes the equation of state for the material using the standard ideal gas equations.

$$P(\rho, E) = \frac{(\gamma - 1)\rho E}{1 - \rho B_v} \tag{12.16}$$

$$E(\rho, T) = C_v T \tag{12.17}$$

where γ is the ratio of constant pressure and constant volume specific heats, C_v is the specific heat, and B_v is the co-volume, representing the volume excluded by molecules of finite size. B_v can be used to make a crude correction for nonideality.

If the material input does not specify the independent variables, the reference conditions in the model input will be used.

- Modules: material_libs/kerley_eos

  - ideal_gas_mig.h
  - ideal_gas_mig.C
  - idgas.doc is the ASCII data file that lists associated Fortran routines.

- Physics: hydrodynamics

Table 38: Input Parameters for `KEOS IDEAL GAS`.

Parameter Name	Type	Description
`GM1`	`real` (2/3)	$\gamma - 1$. Default is monatomic gas value
`CV`	`real`	Specific heat
`BV`	`real` (0.)	Co-volume
`R0` (or `RO`)	`real`	Initial density
`T0` (or `TO`)	`real`	Initial temperature
`E0` (or `EO`)	`real`	Energy at zero pressure and temperature
`TYP`	`real`	Model type (default value set by EOS package). Intended primarily for internal use in EOS package.

Table 39: Registered Plot Variables of `KEOS IDEAL GAS`.

Variable Name	Type	Mode	Description
`DENSITY`	`real`	`INPUT`	Material density
`ENERGY`	`real`	`INPUT`	Specific internal energy
`PRESSURE`	`real`	`OUTPUT`	Pressure
`TEMPERATURE`	`real`	`OUTPUT`	Absolute temperature
`SOUND SPEED`	`real`	`OUTPUT`	Bulk sound speed
`DPDRHO`	`real`	`OUTPUT`	Derivative of pressure with respect to density

```
$ Sample input for keos ideal gas model.
model 141 keos ideal gas
   gm1 = 0.667
   r0  = 1.6245e-04   $ g/cm^3
   t0  = 298.         $ K
   cv  = 3.116324e+07 $ erg/g/K
end
```

12.3.5 KEOS JWL

```
MODEL model_id_number KEOS JWL
   [parameter = value]
   ...
END
```

This is the Kerley EOS version of the Jones-Wilkins-Lee equation of state for high explosives [22]. This model is useful for computing the equation of state of high explosive materials in a fully detonated state. The parameters are unique for a given undetonated density and temperature. Input parameters for this model must be in consistent (`CGS` or `SI`) units, even though JWL parameters are normally published [10] in the following system of units: `A`, `B` and `PCJ` are given in Mbar = 10^{12} GPa, density in gm/cm^3, `DCJ` in cm/μsec, and `E0` in Mbar-cm^3/cm^3.

The setup for this model will fail unless `R0`, `WG`, and either `CV` or `TCJ` are given.

This model can also be used with the `PROGRAMMED BURN` option (Section 8.5.1 on page 177) in ALEGRA. Many of the JWL predefined materials in the `EOS_data` file are assumed to be operating with the `PROGRAMMED BURN` option. To turn off that option, set `BRN` = 0.0.

- Modules: material_libs/kerley_eos
  - jwl_mig.h
  - jwl_mig.C
  - jwl.doc is the ASCII data file that lists associated Fortran routines.
- Physics: hydrodynamics

Table 40: Input Parameters for `KEOS JWL`.

Parameter Name	Type	Description
		continued on next page

continued from previous page

MATLABEL	char	Material label in the EOS_data file. If this parameter is provided, no other parameters are required. Enclose the string in single quotes.
DATAFILE	char	Name and location of the specially formatted EOS_data file if different from the default $ALEGRA_MIGDATA/EOS_data. Enclose the string in single quotes.
R0 (or RO)	real	Density in unreacted reference state (required)
T0 (or TO)	real	Temperature in unreacted reference state (default is 298 K)
AG	real	Constant A in JWL formula in units of problem (required)
BG	real	Constant B in JWL formula in units of problem (required)
R1	real	Dimensionless constant R1 in JWL formula in units of problem (required)
R2	real	Dimensionless constant R2 in JWL formula in units of problem (required)
WG	real	Grüneisen parameter in JWL formula (required)
E0 (or EO)	real	Detonation energy parameter E0 (required for reactive burn). May also be needed to compute specific heat. (If E0 > 0.0, PCJ and DCJ will be recomputed. Otherwise, E0 is computed from DCJ and PCJ if CV = 0.0).
CV	real	Specific heat (if zero, computed from E0 and TCJ)
BRN	real	Set to 0.0 for no heburn with this model. If BRN = 1.0, then the PROGRAMMED BURN input is required.
PCJ	real	Chapman-Jouget pressure. Used only if CV = 0.0 and E0 = 0.0
DCJ	real	Chapman-Jouget detonation front velocity. Used only if CV = 0.0 and E0 = 0.0

continued on next page

continued from previous page		
TCJ	real	Chapman-Jouget temperature. If TCJ = 0.0, TCJ is reset to 0.35 eV or 4061.6 K (used only if CV = 0.0).
TDQ	real (0.)	Delayed heat release (default 0.0 for no time dependent reaction)
TDA	real (0.)	Rate prefactor for time dependent rxn (default 0.0 for no time dependent reaction)
TDM	real (0.)	Exponent for (1.0 - LAMB) (default 0.0 for no time dependent reaction)
TDN	real (0.)	Exponent for P (default 0.0 for no time dependent reaction)

Table 41: Registered Plot Variables of KEOS JWL.

Variable Name	Type	Mode	Description
DENSITY	real	INPUT	Material density
ENERGY	real	INPUT	Specific internal energy
PRESSURE	real	OUTPUT	Pressure
TEMPERATURE	real	OUTPUT	Absolute temperature
SOUND SPEED	real	OUTPUT	Bulk sound speed
DPDRHO	real	OUTPUT	Derivative of pressure with respect to density

Two sample inputs for KEOS JWL model follow.

```
$ This example uses user specified parameters,
$ no programmed burn.

model 151 keos jwl
   r0   = 1.65       $ g/cm^3
   t0   = 298.0      $ K
   ag   = 4.6310e+12 $ dyne/cm^2
   bg   = 8.8730e+10 $ dyne/cm^2
   r1   = 4.550
   r2   = 1.350
   wg   = 0.35
   cv   = 0.         $ calculated internally 9.651e6
```

```
   tcj  = 4061.575   $ K
   e0   = 0.         $ .0745e12
   esft = 0.         $ 1.e10
   dcj  = 7.03e+05   $ cm/s
   pcj  = 2.15E+11   $ dyne/cm^2
   brn  = 0.         $ no heburn
end


$ This example uses the EOS_data file, which assigns
$ BRN = 1 for HMX, so that programmed burn is expected. A
$ separate programmed burn input section is required to
$ define detonation objects, time, and other programmed burn
$ properties.

model 150
   matlabel='HMX'
end
```

12.3.6 KEOS MieGrüneisen

```
MODEL model_id_number KEOS MIEGRUNEISEN
   [parameter = value]
   ...
END
```

The Kerley EOS version of the Mie-Grüneisen equation of state is the most recent version of the MG US UP model in ALEGRA. It is based on the Mie-Grüneisen approximation, with the Hugoniot as the reference curve, together with the expression $\Gamma = \Gamma_0\rho_0/\rho$ for the Grüneisen function. The pressure P and energy E are given by

$$P(\rho, E) = P_H + \Gamma_0\rho_0[E - E_H(\rho)] \tag{12.18}$$

$$E(\rho, T) = E_H + C_v[E - T_H(\rho)] \tag{12.19}$$

where P_H, E_H, and T_H are the Hugoniot pressure, energy, and temperature. The Grüneisen parameter Γ_0 and specific heat C_v are taken to be constants. There are two options for representing the Hugoniot. The first is a quadratic relation between shock velocity U_s and particle velocity U_p,

$$U_s = C_s + S_1U_p + \left(\frac{S_2}{C_s}\right)U_p^2 \tag{12.20}$$

where C_s, S_1, and S_2 are constants. The second option uses a modified form for nonlinear behavior at low pressures.

$$U_s = 2C_s\left[1 - S_1\mu + \sqrt{(1 - S_1\mu)^2 - 4S_2\mu^2}\right]^{-1} - B\exp\left[-(\mu/\mu^*)^N\right] \tag{12.21}$$

This model reduces to the first option if $B = 0$. Both of these options are described in detail by Reference [22].

The `KEOS MieGruneisen` model allows incorporation of the p-alpha porosity model if the appropriate parameters are provided. The p-alpha

model includes a distention parameter, α, which relates the macroscopic material density to the density ρ_M of the solid, void-free material.

$$\alpha = \rho_M / \rho \tag{12.22}$$

The response includes a reversible elastic region and an irreversible compaction region. The maximum distention at a given value of pressure is given by

$$\alpha_{max} = 1 + (\alpha - 1) \left(\frac{P_S - P}{P_S - P_E} \right)^2 \tag{12.23}$$

The parameter P_S is the maximum pressure for complete compaction. All voids are crushed out and $\alpha = 1$ for P greater than or equal to P_S. P_E is the upper pressure limit of the elastic crush region.

The distention parameter, α, is advanced in time using an integration scheme, dividing the global computational time step into several subintervals. The number of subintervals can be adjusted by the user with the NSUB parameter.

The p-alpha model is intended for modeling materials with low porosity (less than 20 percent). It can be used as a submodel with the composite (two-state KEOS models), and is also an option in the `KEOS SESAME` equation of state. For materials with relatively higher porosities, the `KEOS SESAME` model is the recommended choice for the p-alpha option.

- Modules: material_libs/kerley_eos
  - mgrun_mig.h
  - mgrun_mig.C
  - mgrun.doc is the ASCII data file that lists associated Fortran routines.
- Physics: hydrodynamics

Table 42: Input Parameters for KEOS MieGruneisen.

Parameter Name	Type	Description
MATLABEL	char	Material label in the EOS_data file. If this parameter is provided, no other parameters are required. Enclose the string in single quotes.
DATAFILE	char	Name and location of the specially formatted ASCII file which contains the properties for this model. The default is $ALEGRA_MIGDATA/EOS_data. Enclose the string in single quotes.
R0 (or RO)	real	Hugoniot reference density (required).
CS	real	Sound speed in Hugoniot (required).
CV	real	Specific heat (required).
T0 (or TO)	real (298.)	Initial temperature.
S1	real (0.)	Linear coefficient in Hugoniot fit (required).
G0 (or GO)	real (0.)	Grüneisen parameter (required).
S2	real (0.)	Quadratic coefficient in Hugoniot fit.
B	real (0.)	Low-pressure Hugoniot parameter B (default=0 for no low pressure model)
XB	real	Low pressure Hugoniot parameter, required for the low pressure option.
NB	real	Low-pressure Hugoniot parameter N, required for the low pressure option.
RP	real (=R0)	For the p-alpha porosity model[a], initial density of porous material.
PS	real (1.e9)	For the p-alpha porosity model[a], pressure for complete compaction.
PE	real (0.)	For the p-alpha porosity model[a], maximum elastic pressure.
CE	real	For the p-alpha porosity model[a], sound speed in elastic compaction region (default for CE is sound speed of solid matrix).
NSUB	int (10)	For the p-alpha porosity model[a], number of subcycles within time step.

continued on next page

continued from previous page		
ESFT	real	Shift in energy zero (optional).
TYP	real	Model type (default value set by EOS package). Intended primarily for internal use in EOS package (optional).
[a]Providing the parameter RP not equal to R0 designates the use for the p-alpha porosity model. Optional input for the porosity model are the parameters PS, PE, CE, and NSUB.		

Table 43: Registered Plot Variables of KEOS MieGruneisen.

Variable Name	Type	Mode	Description
DENSITY	real	INPUT	Material density
ENERGY	real	INPUT	Specific internal energy per unit mass
PRESSURE	real	OUTPUT	Pressure
TEMPERATURE	real	OUTPUT	Absolute temperature
SOUND SPEED	real	OUTPUT	Bulk sound speed
DPDRHO	real	OUTPUT	Derivative of pressure with respect to density
ALPHA	real	OUTPUT	The porosity parameter if the model parameters describe a porosity model. This parameter represents the ratio solid density/porous density.

```
$ Simplest sample input for \texttt{KEOS MieGruneisen} model.
$ This model uses the p-alpha option since RP and PS are
$ defined in EOS_data, but the value of RP has been adjusted
$ to the solid density (R0 in the EOS_data file) by the user
$ so that the porosity option is not used.

model 131 keos miegruneisen
   matlabel = 'BTF'
   rp       = 1.901   $ turn off p-alpha model.
end
```

12.3.7 KEOS Sesame

```
MODEL model_id_number KEOS SESAME
```

```
   [parameter = value]
   ...
END
```

The KEOS Sesame model is the most up-to-date implementation of the Sesame equation of state in ALEGRA [22]. It uses the most complete and most recent tables, which are identical to the tables used in CTH. For ALEGRA these tables - aneos and sesame - are located in the directory set by the `$ALEGRA_MIGDATA` environment variable. Like most KEOS models, predefined materials are cataloged in the `EOS_data` file. This model allows simplified input in the form of either the `MATLABEL` keyword (followed by a predefined material label listed in the `EOS_data` file) or the `NEOS` keyword (followed by the table number). Other parameters specified in the model input will overwrite those in the standard model.

This model also allows incorporation of the p-alpha porosity model, which was described in the `KEOS MieGruneisen` model (Section 12.3.6 on page 220). The Sesame tabular equation of state provides a more robust equation of state during the compaction of a very porous material and is thus more suitable than the `KEOS MieGruneisen` model for many materials. The user should note that a somewhat smaller time step may be required for very porous materials to facilitate the solution during sudden compaction. The p-alpha model is discussed in greater detail in the reference listed below.

If the `EOS_data` file is used for a predefined material, the user should examine parameters in the `EOS_data` file to see if the porosity model is included (i.e., if `RP`>0). If the porosity model is not desired, set `RP` = `R0` in the model input.

- Modules: material_libs/standard_models
  - sesame_mig.h
  - sesame_mig.C
  - sesame.doc is the ASCII data file that lists associated Fortran routines.
- Physics: hydrodynamics

Table 44: Input Parameters for KEOS SESAME.

Parameter Name	Type	Description
MATLABEL	char	Material label in the EOS_data file. If this parameter is provided, no other parameters are required. Enclose the string in single quotes.
DATAFILE	char	Name and location of the specially formatted ASCII file which contains the properties for this model. The default is $ALEGRA_MIGDATA/EOS_data. Enclose the string in single quotes.
R0 (or RO)	real	Hugoniot reference density.
T0 (or TO)	real	Hugoniot reference temperature.
SR	real (1.0)	Factor for scaling density and energy. Ratio of molecular weight for table to molecular weight of actual material, SR = MW(table) / MW(material).
FEOS	real	Name of file containing table (required). If the name is given with embedded "/" characters, the name is taken literally. Otherwise, the path $ALEGRA_MIGDATA is prepended to the name.
NEOS	real	Number of table for material. Refer to on-line documentation to find material numbers for specific materials (required).
RP	real (=R0)	For the p-alpha porosity model[a], initial density of porous material.
PS	real (1.e9)	For the p-alpha porosity model[a], pressure for complete compaction.
PE	real (0.)	For the p-alpha porosity model, maximum elastic pressure.
CE	real	For the p-alpha porosity model[a], sound speed in elastic compaction region (default for CE is sound speed of solid matrix).

continued on next page

continued from previous page		
NSUB	int (10)	For the p-alpha porosity model[a], number of subcycles within time step.
CLIP	int	If nonzero, the temperature is constrained to lie within the bounds of the underlying table, and the pressure is calculated from this constrained ("clipped") temperature. If zero, no clipping of the temperature is done, and the pressure is extrapolated off the table.
[a]Providing the parameter RP not equal to R0 designates the use for the p-alpha porosity model. Optional input for the porosity model are the parameters PS, PE, CE, and NSUB.		

Table 45: Registered Plot Variables of KEOS SESAME.

Variable Name	Type	Mode	Description
DENSITY	real	INPUT	Material density.
ENERGY	real	INPUT	Specific internal energy per unit mass.
PRESSURE	real	OUTPUT	Pressure.
TEMPERATURE	real	OUTPUT	Absolute temperature.
SOUND SPEED	real	OUTPUT	Bulk sound speed.
DPDRHO	real	OUTPUT	Derivative of pressure with respect to density.
ALPHA	real	OUTPUT	Porosity parameter if p-alpha model used.

```
$ Sample input for keos sesame with p-alpha model
$ included. Table 4020 is in the table names 'aneos', which is
$ located in the $ALEGRA_MIGDATA directory.
model 191 keos sesame
   neos = 4020
   feos = 'aneos'
   r0   = 2.785
   rp   = 2.15
   pe   = 4.5e8
end
```

12.3.8 MG Power

```
MODEL model_id_number MG POWER
   [parameter = value]
   ...
END
```

This a general Mie-Grüneisen power law equation of state form generally applicable to solid equations of state but with a fair amount of flexibility on the actual shape of the reference curves. It is assumed that the pressure is related to the density and specific energy through

$$P(\rho, E) = P_R(\rho) + \Gamma\rho\left[E - E_R(\rho)\right] \tag{12.24}$$

and

$$E(\rho, T) = E_R(\rho) + C_v\left[T - T_R(\rho)\right] \tag{12.25}$$

where

$$\Gamma\rho = \Gamma_0\rho_0 \tag{12.26}$$

and C_v are constants. The subscript R refers to a reference state curve which can be an isentrope or Hugoniot for example. This particular model utilizes three regions: a compressive region, a tension region, and a "fracture" region as described below. We define the volumetric strain η as

$$\eta = 1 - \frac{\rho_0}{\rho} = 1 - \frac{\nu}{\nu_0} \tag{12.27}$$

For $\eta > 0$ we define:

$$P_R = P_H = K_0\eta\left(1 + K_1\eta + K_2\eta^2 + K_3\eta^3 + K_4\eta^4 + K_5\eta^5\right) \tag{12.28}$$

where

$$K_0 = \rho_0 C_0^2 \tag{12.29}$$

defines the bulk modulus.

$$E_R = E_H = \frac{P_H \eta}{2\rho_0} + E_0 \tag{12.30}$$

Using the method of Walsh and Christian [43] it can be shown that the temperature on the Hugoniot curve is given by

$$T_H = T_0 e^{\Gamma_0 \eta} + \frac{\Gamma_0 \eta}{2C_v \rho_0} \int_0^\eta e^{-\Gamma_0 z} z^2 \frac{d}{dz}\left(\frac{P_H}{z}\right) dz \tag{12.31}$$

which leads to

$$T_H = T_0 e^{\Gamma_0 \eta} + \frac{\Gamma_0 \eta}{2C_v \rho_0} K_0 \left(K_1 I_2 + 2K_2 I_3 + 3K_3 I_4 + 4K_4 I_5 + 5K_5 I_6\right) \tag{12.32}$$

where

$$I_n = \int_0^\eta e^{-\Gamma_0 z} z^n dz \tag{12.33}$$

The associated incomplete gamma function is evaluated as a recursion for $\Gamma_0 \eta \geq 1$ and as a series otherwise for accuracy purposes.

For $\eta < 0$ we define the reference curve by an isentrope:

$$P_R = P_{ISEN} = K_0 \eta \tag{12.34}$$

$$E_R = E_{ISEN} = \frac{K_0 \eta^2}{2\rho_0} + E_0 \tag{12.35}$$

$$T_R = T_{ISEN} = T_0 e^{\Gamma_0 \eta} \tag{12.36}$$

For $\eta < \eta_{min} = \frac{P_{min}}{K_0}$ we have:

$$P_R = P_{ISEN} = P_{min} \tag{12.37}$$

$$E_R = E_{ISEN} = \frac{K_0 \eta_{min}^2}{2\rho_0} + E_0 + \frac{P_{min}}{\rho_0}(\eta - \eta_{min}) \tag{12.38}$$

$$T_R = T_{ISEN} = T_0 e^{\Gamma_0 \eta} \tag{12.39}$$

In all cases the sound speed is given by:

$$C_s^2 = \nu^2 \left[\rho_0 P_R'(\eta) - \rho_0 \Gamma_0 \left(-P + \rho_0 E_R'(\eta)\right)\right] \tag{12.40}$$

- Modules: material_libs/standard_models
  - mgpower.h
  - mgpower.C
- Physics: hydrodynamics

Example:

```
model 61 mg power
   rho ref = 2.37     $ g/cm^3
   tref    = 273.     $ K
   gamma0  = 1.
   cv      = 1.5e7    $ erg/g/K
   k0      = 1.96e11  $ dyne/cm^2
   k1      = -4.9
   k2      = 31.0
end
```

Table 46: Input Parameters for MG POWER.

Parameter Name	Type	Description
RHO REF	real	Material density at the reference state (required).
TREF	real	Absolute temperature at the reference state (T_0).
CV	real	Specific heat at constant volume (C_v).
E SHIFT	real	Arbitrary energy shift (E_0) (optional).
GAMMA0	real	Grüneisen parameter (Γ_0).
C0	real	Reference bulk wave speed (C_0). If C0 = 0, the bulk modulus will be used to calculate the wave speed: C0 = sqrt(K0/RHO_REF). Either C0 or K0 must be greater than 0; the other must be zero.
K0	real	Bulk modulus (K_0). If K0 = 0 and C0 > 0, the wave speed will be used to calculate the bulk modulus: K0=RHO_REF*C0*C0. Either C0 or K0 must be greater than 0; the other must be zero.
K1	real	Hugoniot coefficient (K_1).
K2	real	Hugoniot coefficient (K_2).
K3	real	Hugoniot coefficient (K_3).
K4	real	Hugoniot coefficient (K_4).
K5	real	Hugoniot coefficient (K_5).
PRESSURE CUTOFF	real	

12.3.9 MG US UP

```
MODEL model_id_number MG US UP
   [parameter = value]
   ...
END
```

Mie-Grüneisen Us-Up equation of state is applicable to high compression of metal solids [27]. It provides a convenient way to define an equation of state from shock Hugoniot data. This equation of state model assumes a linear

Table 47: Registered Plot Variables of MG POWER.

Variable Name	Type	Mode	Description
SPECIFIC HEAT VOL	real	INITIALIZE	Initialized to CV parameter.
DENSITY	real	INPUT	Material density.
ENERGY	real	INPUT	Specific internal energy per unit mass.
PRESSURE	real	OUTPUT	Pressure.
TEMPERATURE	real	OUTPUT	Absolute temperature.
SOUND SPEED	real	OUTPUT	Bulk sound speed.

relation between shock velocity, U_s, and particle velocity, U_p, according to:

$$U_s = c_0 + sU_p \tag{12.41}$$

The Grüneisen parameter Γ is a function of density according to:

$$\Gamma(\rho) = \Gamma_0 \frac{\rho_0}{\rho} \tag{12.42}$$

and the material pressure, internal energy and temperature are related to the reference state by:

$$P(\rho, E) = P_R + \Gamma_0 \rho_0 [E - E_R(\rho)] \tag{12.43}$$

$$E(\rho, T) = E_R + C_v [E - T_R(\rho)] \tag{12.44}$$

where P_R, E_R, and T_R are analytic functions of the density. The above formulation can give misleading results when $\rho < \rho_0$, thus the model has been modified to use an "expansion equation of state" when the material goes into tension. This model was implemented into an early version of CTH[21] where the particular forms of the reference curves used in this model are documented.

- Modules: material_libs/standard_models

  - matmod_mgusup.h
  - matmod_mgusup.C
- Physics: hydrodynamics

Table 48: Input Parameters for `MG US UP`.

Parameter Name	Type	Description
`SL`	`real`	Linear constant s in Us-Up equation
`C0`	`real`	Constant c_0 in Us-Up equation
`GAMMA0`	`real`	Grüneisen parameter, Γ_0
`RHO REF`	`real`	Material density at reference state, ρ_0
`EREF`	`real`	Internal energy at reference state, E_R
`PREF`	`real`	Pressure at reference state, P_R
`TREF`	`real`	Absolute temperature at reference state, T_R
`CV`	`real`	Specific heat at constant volume
`E SHIFT`	`real`	Arbitrary energy shift (optional)
`PRESSURE CUTOFF`	`real`	Minimum pressure allowed

Table 49: Registered Plot Variables of `MG US UP`.

Variable Name	Type	Mode	Description
`SPECIFIC HEAT VOL`	`real`	`INITIALIZE`	Initialized from `CV` parameter
`DENSITY`	`real`	`INPUT`	Material density
`ENERGY`	`real`	`INPUT`	Specific internal energy per unit mass
`PRESSURE`	`real`	`OUTPUT`	Pressure
`TEMPERATURE`	`real`	`OUTPUT`	Absolute temperature
`SOUND SPEED`	`real`	`OUTPUT`	Bulk sound speed
`DPDRHO`	`real`	`OUTPUT`	Derivative of pressure with respect to density

```
$ Sample input for mg us up.
$ KEOS MieGruneisen will eventually supersede this model.

model 112 mg us up
```

```
   c0      = 3.94e5   $ cm/s
   sl      = 1.489
   gamma0  = 1.99
   rho ref = 8.93     $ g/cm^3
   cv      = 3.929e06 $ erg/g/K
   pref    = 0.0
   tref    = 298.     $ K
end
```

12.4 Constitutive Models

The constitutive models define the equations for the stress and deformation relationships in a material. This section focuses on the basic constitutive models in ALEGRA. A later section provides descriptions of models that may include the constitutive relations with more complex yield behavior.

Let us begin with one comment regarding the use of Young's modulus in the following models. Young's modulus is generally an unfamiliar elastic constant for people used to working with shock waves. Rather, they are used to working with the bulk modulus. Below, we indicate the relationship between these two moduli, given the Poisson ratio .

The bulk modulus is defined by:

$$B_0 = \rho_0 C_0^2 \tag{12.45}$$

where the subscript "0" refers to reference conditions, B_0 is the bulk modulus, ρ_0 is the density, and C_0 is the sound speed. The bulk modulus is easily computed for most materials from this formula since sound speed and density are almost always known at reference conditions for the shock wave applications we are interested in. If E is the Young's modulus, then it can be found from the bulk modulus by the following formula:

$$E_0 = 3(1-2\nu)B_0 = 3(1-2\nu)\rho_0 C_0^2 \tag{12.46}$$

12.4.1 Elastic Plastic

```
MODEL model_id_number ELASTIC PLASTIC
   [parameter = value]
   ...
END
```

This is a classical elastic plastic constitutive model using a generalized Hooke's Law for elastic stress-strain response, von Mises yield criteria, combined isotropic and kinematic hardening, and simple radial return [9, 38]. The ALEGRA implementation has been adapted from the elastic plastic model implemented in PRONTO3D [39]. The hardening mode is weighted by the input parameter, `BETA`, where 0 is fully kinematic, and 1 is fully isotropic hardening.

This model also has a pressure-dependent yield strength extension. This extension follows the development of the GEO strength model in CTH. The radius of the yield surface is calculated as a von Mises yield criteria. If pressure-dependence is indicated, the yield strength is modified by:

$$Y = \left[\frac{Y_{max}}{Y_0} - \left(\frac{Y_{max}}{Y_0} - 1\right) \exp\left(\frac{PY_s}{Y_{max} - Y_0}\right)\right] Y_{np} \tag{12.47}$$

where Y_{np} is the pressure-independent yield strength, Y_{max} and Y_0 are the asymptotic and zero-pressure values of the yield stress in uniaxial tension, respectively. The parameter Y_s is the slope of the yield stress at zero pressure. If $Y_0 \geq Y_{max}$, then the pressure dependence factor calculation is skipped.

Due to input parsing issues, this model is often inadvertently used instead of the `LINEAR ELASTIC` model. For example, a user, intending to apply the `LINEAR ELASTIC` material model to a material might provide the following input:

```
model 121 elastic
   youngs modulus = 1.e12
   poissons ratio = 0.3
end
```

In this case, the model assumes that a linear elastic material has been specified and sets the initial yield stress to a large number to ensure no yield occurs.

- Modules: material_libs/standard_models
  - matmod_ep.h
  - matmod_ep.C
- Physics: solid dynamics

```
$ Sample input for elastic plastic model

model 31 elastic plastic
```

Table 50: Input Parameters for ELASTIC PLASTIC.

Parameter Name	Type	Description
YOUNGS MODULUS	real	Young's Modulus of the material
POISSONS RATIO	real	Poisson's ratio of the material
YIELD STRESS	real	Initial yield stress in uniaxial tension
HARDENING MODULUS	real	Hardening modulus for yield stress
BETA	real	Weight for kinematic/isotropic hardening 0 = fully kinematic, 1 = fully isotropic).
YIELD MAX	real	Asymptotic yield stress for pressure dependence
YIELD SLOPE	real	Slope of yield stress at zero pressure

```
   youngs modulus    = 1.076e+12  $ dyne/cm^2
   poissons ratio    = 0.355
   yield stress      = 6.0e+09    $ dyne/cm^2
   hardening modulus = 2.0e+09    $ dyne/cm^2
   beta              = 1.0
end
```

12.4.2 Linear Elastic

```
MODEL model_id_number LINEAR ELASTIC
   [parameter = value]
   ...
END
```

This model [38] computes the stress tensor using an incremental, generalized Hooke's Law.

$$d\sigma_{ij} = \lambda d\epsilon_{kk}\delta_{ij} + 2\mu d\epsilon_{ij} \tag{12.48}$$

where σ and ϵ are the stress and strain, respectively, and λ and μ are Lame constants. The incremental strain is obtained by integrating the deformation rate over the time interval of the step.

Table 51: Registered Plot Variables of ELASTIC PLASTIC.

Variable Name	Type	Mode	Description
BULK_MODULUS	real	INITIALIZE	Bulk modulus
SHEAR_MODULUS	real	INITIALIZE	Shear modulus
DEFORMATION_RATE	symtensor	INPUT	Rate of deformation tensor
STRESS	symtensor	IOPUT	Cauchy stress tensor
BACK_STRESS	symtensor	IOPUT	Cauchy back stress tensor for kinematic hardening
EQPS	real	IOPUT	Equivalent plastic strain
YIELD_STRESS	real	IOPUT	Yield stress in uniaxial tension

- Modules: material_libs/standard_models
  - matmod_elastic.h
  - matmod_elastic.C
- Physics: solid dynamics

Table 52: Input Parameters for LINEAR ELASTIC.

Parameter Name	Type	Description
YOUNGS MODULUS	real	Young's Modulus of the material
POISSONS RATIO	real	Poisson's ratio of the material

```
$ Sample input for linear elastic model.

model 41 linear elastic
   youngs modulus = 1.076e+12  $ dyne/cm^2
   poissons ratio = 0.355
end
```

Table 53: Registered Plot Variables of `LINEAR ELASTIC`.

Variable Name	Type	Mode	Description
BULK_MODULUS	real	INITIALIZE	Bulk modulus
SHEAR_MODULUS	real	INITIALIZE	Shear modulus
DEFORMATION_RATE	symtensor	INPUT	Rate of deformation tensor
STRESS	symtensor	IOPUT	Cauchy stress tensor

12.4.3 Soil Crushable Foam

```
MODEL model_id_number SOIL CRUSHABLE FOAM
   [parameter = value]
   ...
END
```

This is an implementation of the PRONTO soil and crushable foam model [38]. This model has a pressure-volumetric strain response superposed on a basic elastic plastic response. The pressure-volumetric strain controls pressure lockup, unload, fracture, and reloading response of the material.

- Modules: material_libs/standard_models
  - matmod_soil_foam.h
  - matmod_soil_foam.C
- Physics: solid dynamics

```
$ sample input for soil crushable foam model.
model 51 soil crushable foam
   shmod = 1.4e10
   bulk  = 2.76e11
   a0    = 398083.
   a1    = 0.019245
   a2    = -1.163e-10
   pfrac = -18595369.
   pmax  = 82738607.
```

Table 54: Input Parameters for SOIL CRUSHABLE FOAM.

Parameter Name	Type	Description
SHMOD	real	Shear modulus
BULK	real	Bulk modulus
A0	real	Constant parameter for P-v curve
A1	real	Linear parameter for P-v curve
A2	real	Quadratic parameter for P-v curve
PFRAC	real	Fracture pressure or tensile limit
PMAX	real	Not input; normally set to -A1/(2.*A2), or a large number if A2 = 0. If the pressure > PMAX, the yield stress is set using PMAX.
FUNCTION TABLE	real	Function table id for an input P-v curve

Table 55: Registered Plot Variables of SOIL CRUSHABLE FOAM.

Variable Name	Type	Mode	Description
DEFORMATION_RATE	symtensor	INPUT	Deformation rate tensor
STRESS	symtensor	IOPUT	Cauchy stress tensor
EV	state	IOPUT	Volumetric strain
EVFRAC	state	IOPUT	Volumetric strain at fracture
EVMAX	state	IOPUT	Maximum previous volumetric strain
NUM	state	IOPUT	Last increment in pressure function where interpolate was found.

```
  func table = 51
end

function 51  $ P-mu curve for soil material
$ mu       P
  0.00000  0.00000e+00
  0.00500  9.38215e+07
  0.01000  1.88356e+08
  0.01500  2.95625e+08
     $ (data omitted in sample for the sake of brevity)
  0.51500  1.15825e+11
```

```
   0.52000  1.17203e+11
end
```

12.4.4 Isotropic Geomaterial

```
MODEL model_id_number ISOTROPIC GEOMATERIAL
   [parameter = value]
   ...
END
```

This model is the implementation of the Fossum isotropic geomaterial model [14, 13] for use in geologic materials in the elastic to moderate strain rate range. It is described as a unified, compaction/dilatation, continuous-surface, strain-rate sensitive, plasticity model. It uses the three stress invariants. Isotropic hardening or softening is a function of volumetric strain and kinematic hardening or softening is a function shear strain. It uses an associative flow.

- Modules: material_libs/isotropic-geomaterial
  - Isotropic_Geomaterial.h
  - Isotropic_Geomaterial.C
  - Isotropic_Geomaterial_init.F
  - Isotropic_Geomaterial_calcs.F
  - geochk.F
- Physics: solid dynamics

Table 56: Input Parameters for `ISOTROPIC GEOMATERIAL`.

Parameter Name	Type	Description
`B0`	`real`	Initial elastic bulk modulus (stress)
`B1`	`real`	High pressure coefficient in nonlinear elastic bulk modulus function (stress)

continued on next page

continued from previous page

B2	real	Curvature parameter in nonlinear elastic bulk modulus function (stress)
B3	real	Coefficient in nonlinear elastic bulk modulus to allow for plastic softening (stress)
B4	real	Power in bulk modulus softening (dimensionless)
G0	real	Initial elastic shear modulus (stress)
G1	real	Coefficient in shear modulus hardening (dimensionless)
G2	real	Curvature parameter in shear modulus hardening (1/stress)
G3	real	Coefficient in shear modulus softening (stress)
G4	real	Power in shear modulus softening (dimensionless)
RJS	real	Joint spacing (length)
RKS	real	Joint shear stiffness (stress)
RKN	real	Joint normal stiffness (stress)
A1	real	Constant term for meridional profile function of ultimate shear limit surface (stress)
A2	real	Curvature decay parameter in the meridional profile function (1/stress)
A3	real	Parameter in the meridional profile function (stress^2)
A4	real	High-pressure slope parameter in merdional profile function (dimensionless)
P0	real	One third of the elastic limit pressure parameter at onset of pore collapse (stress)
P1	real	One third of slope of porosity vs pressure crush curve at elastic limit (1/stress)
P2	real	Parameter for hydrostatic crush curve (1/stress^2)
P3	real	Asymptote of the plastic volumetric strain for hydrostatic crush (dimensionless)
CR	real	Parameter for porosity affecting shear strength (dimensionless)
RK	real	Triaxial extension strength to compression strength ratio (dimensionless)

continued on next page

continued from previous page		
RN	real	Initial shear yield offset (stress)
HC	real	Kinematic hardening parameter (stress)
CUTI1	real	Tension cut-off value of I1 (stress)
CUTPS	real	Tension cut-off value of principal stress (stress)
T1	real	Relaxation time constant 1 (time)
T2	real	Relaxation time constant 2 (strain-rate)
T3	real	Parameter no longer in use
T4	real	Parameter no longer in use
T5	real	Relaxation time constant 5 (stress)
T6	real	Relaxation time constant 6 (time)
T7	real	Relaxation time constant 7 (1/stress)
J3TYPE	int	Type of 3^{rd} deviatoric stress invariant function 1 = Gudehus (default), 2 = Willam-Warnke, 3 = Mohr-Coulomb
A2PF	real	Potential function parameter 1 (1/stress) (default=A2)
A4PF	real	Potential function parameter 2 (radians) (default=A4)
CRPF	real	Potential function parameter 3 (dimensionless) (default=CR)
RKPF	real	Potential function parameter 4 (dimensionless) (default=RK)
SUBX	real	Subcycle control parameter (dimensionless)

Table 57: Registered Plot Variables for ISOTROPIC GEOMATERIAL.

Variable Name	Type	Mode	Description
DEFORMATION_RATE	symtensor	INPUT	Deformation rate tensor
STRESS	symtensor	IOPUT	Cauchy stress tensor
KAPPA	real	IOPUT	Isotropic hardening variable
INDEX	real	OUTPUT	Indicator for plastic hardening
EQDOT	real	OUTPUT	L2 Norm of input strain rate
I1	real	OUTPUT	First invariant of stress
ROOTJ2	real	OUTPUT	Second invariant of deviatoric stress
ALPHA	symtensor	IOPUT	Back stress components
			continued on next page

continued from previous page			
GFUN	real	IOPUT	Kinematic hardening decay function
EQPS	real	IOPUT	Equivalent plastic shear strain
EQPV	real	IOPUT	Equivalent plastic volume strain
EL0	real	IOPUT	Calculated initial value of EL
BACKRN	real	OUTPUT	Second invariant of back stress
CRACK	real	OUTPUT	= 1.0 for tensile cracking failure
SHEAR	real	OUTPUT	= 1.0 for shear failure
YIELD	real	OUTPUT	Yield function (<= 1.0)
LODE	real	OUTPUT	Angle (in degrees) to the stress point in the pi-plane, extension = -30.0, pure shear = 0.0, compression = 30.0

```
$ Sample input for isotropic geomaterial model.

  model 400 isotropic geomaterial
    B0 = 3.333000e+10  $Pa
    B1 = 42469.7e6   $Pa
    B2 = 410.7e6     $Pa
    B3 = 12000.e6    $Pa
    B4 = 0.021       $Dimensionless
    G0 = 1.27800e+10 $Pa
    G1 = 0.0         $Dimensionless
    G2 = 0.0         $1/Pa
    G3 = 0.0         $Pa
    G4 = 0.0         $Dimensionless
    RJS = 1.2700e-02
    RKS = 2.000e+11
    RKN = 1.000e+10
    A1 = 843.02e6    $Pa
    A2 = 2.731e-10   $1/Pa
    A3 = 821.92e6    $Pa
    A4 = 1.e-10      $Radians
    P0 = -314.4e6    $Pa
    P1 = 1.22e-10    $1/Pa
    P2 = 1.28e-18    $1/Pa^2
    P3 = 0.135       $ [-]
    CR = 6.0         $Dimensionless
```

```
  RK = .72          $Dimensionless
  RN = 0.0000e+00
  HC = 0.0000e+00
  CTI1 = 3.e6       $Pa
  CTPS = 1.e6       $Pa
  T1 = 4.7e-4       $sec
  T2 = 0.810        $1/sec
  T3 = 0.0          $Dimensionless
  T4 = 0.0          $1/sec
  T5 = 0.0          $Pa
  T6 = 3.5          $sec
  T7 = 0.0          $1/Pa
  J3TYPE = 3        $Dimensionless
  A2PF = 0.0        $ A2PF defaults to A2 for normality
  A4PF = 0.0        $ A4PF defaults to A4 for normality
  CRPF = 0.0        $ CRPF defaults to CR for normality
  RKPF = 0.0        $ RKPF defaults to RK for normality
end
```

12.5 Yield Models

To simplify input for several of the yield models, a specially formatted ASCII file named `VP_data` is distributed with ALEGRA that provides input parameters for many predefined materials. Incorporation of any particular data set into the `VP_data` file does not imply correctness or validation of the data set, but it is intended as a starting point for materials of interest in the absence of experimental data for the particular application. Parameters in the `VP_data` file are given in `CGSEV` units (see Section 4.1.3 on page 69), which are standard CTH units (the `VP_data` file is also distributed with CTH). The data are read and automatically converted to problem units during ALEGRA's set up phase. The default location of the `VP_data` file is set by the environment variable `$ALEGRA_MIGDATA`. All models in the `VP_data` file are not necessarily accessible by the ALEGRA code, and conversely, all models in the ALEGRA code do not have data provided in the `VP_data` file. For applicability to a specific model, look in the input data below for the `MATLABEL` input.

12.5.1 Steinberg-Guinan-Lund

This model is the MIG implementation of the Steinberg-Guinan-Lund model [40] which is also used in the CTH code. This model must be used as the yield stress submodel to the `CTH ELASTIC PLASTIC` model so that the equation of state, the yield model, and the radial return algorithm are properly coupled (see 12.7 on 260).

The model and its implementation is described fully by Taylor, but the basic equations will be outlined here as a brief introduction. The Steinberg-Guinan-Lund model predicts the viscoplastic response of various materials (principally metals) based on a consideration of thermally-activated dislocation mechanics. The strain rate dependent form of the SGL model defines the yield stress as

$$Y = \left[Y_T(\dot{\epsilon}, T) + Y_A f(\epsilon^p)\right] \frac{G(P,T)}{G_0} \tag{12.49}$$

where the athermal and thermally activated components $Y_A f(\epsilon^p)$ and Y_T are

defined by

$$Y_A f(\epsilon^p) = Y_A \left[1 + \beta(\epsilon^p + \epsilon_i)\right]^n \tag{12.50}$$

$$\dot{\epsilon}^p = \left(\frac{1}{C_1} \exp \left[\frac{2U_K}{T} \left(1 - \frac{Y_T}{Y_P} \right)^2 \right] \frac{C_2}{Y_T} \right)^{-1} \tag{12.51}$$

In these equations, P and T are the pressure and temperature, ϵ^p and $\dot{\epsilon}^p$ are the equivalent plastic strain and its time derivative, Y_A is the yield strength at the Hugoniot elastic limit, $f(\epsilon^p)$ is the work-hardening function with β, ϵ, and n used as fitting parameters. G_0 is the initial shear modulus, Y_P is the Peierls stress, and $2U_K$ is the energy necessary to form a pair of kinks in a dislocation segment. The quantities C_1 and C_2 are defined in terms of various dislocation mechanics parameters and are specific to the material being modeled. Two limits are imposed in this model.

$$Y_A f(\epsilon^p) \leq Y_{max} \quad \text{and} \quad Y_T \leq Y_P \tag{12.52}$$

where Y_{max} is the work-hardening maximum in the rate-dependent version of the model.

The rate-independent model is a special case of the rate-dependent form with Y_T set to zero and the following limit applied:

$$Y_0 f(\epsilon^p) \leq Y_{max} \tag{12.53}$$

- Modules: material_libs/steinberg-guinan-lund
  - st_mig.h
  - st_mig.C
  - st.dat is the ASCII data file that lists the associated Fortran routines.
- Physics: solid dynamics

Table 58: Input Parameters for `STEINBERG GUINAN LUND`.

Parameter Name	Type	Description
`MATLABEL`	`string`	Material label in the `VP_data` file. If this parameter is provided, no other parameters are required. Enclose the string in single quotes.
`DATAFILE`	`string`	Name and location of the specially formatted `VP_data` file if different from the default `$ALEGRA_MIGDATA/VP_data`. Enclose the string in single quotes.
`ROST`	`real`	Initial density ρ_0
`TMOST`	`real`	Melting temperature T_m
`ATMST`	`real`	Material constant a in Lindemann melting law
`GMOST`	`real`	Initial Grüneisen coefficient γ_0
`AST`	`real`	Material constant A in definition of shear modulus
`BST`	`real`	Material constant B in definition of shear modulus
`NST`	`real`	Parameter n in work hardening function f
`C1ST`	`real`	Material constant C_1 for thermal yield stress
`C2ST`	`real`	Material constant C_2 for thermal yield stress
`GOST`	`real`	Initial shear modulus G_0
`BTST`	`real`	Parameter β in work-hardening function f
`EIST`	`real`	Initial equivalent plastic strain ϵ_i
`YPST`	`real`	Peierls stress and maximum value of Y_T
`UKST`	`real`	Activation energy
`YSMST`	`real`	Max yield stress of athermal yield component Y_{max}
`YAST`	`real`	Athermal yield stress prefactor Y_A
`YOST`	`real`	Initial yield stress for rate-independent version Y_0
`YMST`	`real`	Max yield stress for rate-independent version Y_{max}

Table 59: Registered Plot Variables for `STEINBERG GUINAN LUND`

Variable Name	Type	Mode	Description
DEFRATE	symtensor	INPUT	Rate of deformation tensor
MATFRAC	real	INPUT	Volume fraction of material
TEMPERATURE	real	INPUT	Absolute temperature
DENSITY	real	INPUT	Material density
PRESSURE	real	INPUT	Pressure
EQPS	real	INPUT	Equivalent plastic strain (previous cycle)
STRESS	real	INPUT	Cauchy stress (deviator from previous cycle, mean stress is current pressure)
YIELD_STRESS	real	OUTPUT	Yield stress in uniaxial tension
SHEAR_MODULUS	real	OUTPUT	Shear modulus

```
$ Sample input for steinberg guinan lund yield model.

model 103 steinberg guinan lund
   r0st  = 16.69
   tm0st = 4340.
   atmst = 1.30
   gm0st = 1.67
   ast   = 1.45e-12
   bst   = 1.508650
   nst   = 0.10
   c1st  = 0.71e+06
   c2st  = 0.12e+06
   g0st  = 0.690e+12
   btst  = 10.0
   eist  = 0.00
   ypst  = 8.2e+09
   ukst  = 0.31
   ysmst = 4.5e+09
   yast  = 3.75e+09
   y0st  = 7.7e+09
   ymst  = 1.1e+10
end
```

12.5.2 Johnson-Cook EP

This model is the MIG implementation of the Johnson-Cook viscoplastic model [30]. This MIG module is also used in the CTH code. In ALEGRA this model must be used as the yield-stress submodel to the `CTH ELASTIC PLASTIC` model so that the equation of state, the yield model, and the radial return algorithm are properly coupled (see 12.7 on 260).

In this model the yield stress is dependent on temperature, rate of deformation, and history of deformation. The governing equation has the following form:

$$Y = \left[A + B(\epsilon^p)^N\right]\left[1 + C\ln\left(\max(0.002, \dot{\epsilon}^p)\right)\right]\left[1 - \theta_h^m\right] \tag{12.54}$$

where A, B, C, N, and m are constants that depend on the material, and $\dot{\epsilon}^p$ is the time derivative of the equivalent plastic strain. θ_h is the homologous temperature, defined by

$$\theta_h = \frac{T - T_r}{T_M - T_r} \tag{12.55}$$

where T_r and T_M are room temperature and the material melting temperatures. Reference [30] describes the implementation of the equations in detail and also provides the parameters for 12 materials.

- Modules: material_libs/johnson_cook_ep
  - jcep_mig.h
  - jcep_mig.C
  - jcep.dat is the ASCII data file that lists the Fortran files.
- Physics: solid dynamics

```
$ Sample input for Johnson-Cook model for copper.

model 113 johnson cook ep
   ajo = 8.970000e+08
```

Table 60: Input Parameters for JOHNSON COOK EP.

Parameter Name	Type	Description
MATLABEL	string	Material label in the VP_data file. If this parameter is provided, no other parameters are required. Enclose the string in single quotes.
DATAFILE	string	Name and location of the specially formatted VP_data file if different from the default $ALEGRA_MIGDATA/VP_data. Enclose the string in single quotes.
AJO	real	Material constant A
BJO	real	Material constant B
CJO	real	Material constant C
MJO	real	Material constant m
NJO	real	Material constant N
TJO	real	Material melting temperature T_m

```
   bjo = 2.918700e+09
   cjo = 2.500000e-02
   mjo = 1.090000e+00
   njo = 3.100000e-01
   tjo = 1.189813e-01
end
```

12.5.3 Zerilli-Armstrong

This model is the MIG implementation of the Zerilli-Armstrong viscoplastic model [30]. This MIG module is also used in the CTH code. In ALEGRA, the Zerilli-Armstrong model must be used as the yield stress submodel to the CTH ELASTIC PLASTIC model so that the equation of state, the yield model, and the radial return algorithm are properly coupled (see 12.7 on 260).

Like the Johnson-Cook model, in this model the yield stress is dependent on temperature, rate of deformation, and history of deformation. The Zerilli-Armstrong model is based on a physical model of the crystal structure of the

Table 61: Registered Plot Variables for JOHNSON COOK EP.

Variable Name	Type	Mode	Description
DEFRATE	symtensor	INPUT	Rate of deformation tensor
MATFRAC	real	INPUT	Volume fraction of material
TEMPERATURE	real	INPUT	Absolute temperature
EQPS	real	INPUT	Equivalent plastic strain (previous cycle)
STRESS	real	INPUT	Cauchy stress (deviator from previous cycle, mean stress is current pressure)
SHEAR_MODULUS	real	INPUT	Shear modulus
YIELD_STRESS	real	OUTPUT	Yield stress in uniaxial tension

material. The simplified governing equation has the following form:

$$Y = \nabla\sigma'_G + k\sqrt{(l)} + \left(c_1 + c_2\sqrt{\epsilon^p}\right)\exp\left(-c_3 T + c_4 T\ln(\dot{\epsilon}^p)\right) + c_5(\epsilon^p)^N \quad (12.56)$$

where the parameters are defined in the table below, T is temperature, and ϵ^p is the equivalent plastic strain. Reference [30] describes the implementation of the equation in detail and also provides the parameters for copper and iron.

- Modules: material_libs/zerilli_armstrong
  - za_mig.h
  - za_mig.C
  - za.dat is the ASCII data file for further information.
- Physics: solid dynamics

```
$ Sample input for zerilli armstrong

model 123 zerilli armstrong
   c1ze = 0.000000e+00
```

Table 62: Input Parameters for `ZERILLI ARMSTRONG`.

Parameter Name	Type	Description
`MATLABEL`	`string`	Material label in the `VP_data` file. If this parameter is provided, no other parameters are required. Enclose the string in single quotes.
`DATAFILE`	`string`	Name and location of the specially formatted `VP_data` file if different from the default `$ALEGRA_MIGDATA/VP_data`. Enclose the string in single quotes.
`C1ZE`	`real`	Material constant c_1 (units of pressure)
`C2ZE`	`real`	Material constant c_2 (units of pressure)
`C3ZE`	`real`	Material constant c_3 (units of temperature^{-1})
`C4ZE`	`real`	Material constant c_4 (units of temperature^{-1})
`C5ZE`	`real`	Material constant c_5 (units of pressure)
`AZE`	`real`	Material constant $\nabla\sigma'_G + k\sqrt{(l)}$ (units of pressure)
`NZE`	`real`	Material constant N (dimensionless)

```
   c2ze = 8.900000e+09
   c3ze = 2.80e-3
   c4ze = 1.15e-4
   c5ze = 0.000000e+00
   aze  = 6.500000e+08
   nze  = 1.000000e+00
end
```

12.5.4 Bammann-Chiesa-Johnson

This model is the MIG implementation of the Bammann-Chiesa-Johnson viscoplastic damage model [41] which is also used in the CTH code. Taylor describes the model and implementation fully and also provides the parameters for five metals.

The viscoplastic component of this model incorporates isotropic and kinematic hardening as well as strain rate and thermal effects. Damage modeling is based on an analytic expression for spherical void growth. The damage growth rate is dependent on the effective stress, tensile pressure, plastic strain

Table 63: Registered Plot Variables for ZERILLI ARMSTRONG.

Variable Name	Type	Mode	Description
DEFRATE	symtensor	INPUT	Rate of deformation tensor
MATFRAC	real	INPUT	Volume fraction of material
TEMPERATURE	real	INPUT	Absolute temperature
EQPS	real	INPUT	Equivalent plastic strain (previous cycle)
STRESS	real	INPUT	Cauchy stress (deviator from previous cycle, mean stress is current pressure)
SHEAR_MODULUS	real	INPUT	Shear modulus
YIELD_STRESS	real	OUTPUT	Yield stress in uniaxial tension

rate, and the current damage level. The basic equations are outlined below.

A linear elastic response is represented by

$$\hat{\tilde{\sigma}} = \lambda(1-\phi)tr(\tilde{D}^e)\tilde{1} + 2\mu(1-\phi)\tilde{D}^e - \frac{\dot{\phi}}{(1-\phi)}\tilde{\sigma} \tag{12.57}$$

where the Cauchy stress $\tilde{\sigma}$ is convected with the spin $\tilde{W}$ according to the Jaumann rate

$$\hat{\tilde{\sigma}} = \dot{\tilde{\sigma}} - \tilde{W}\tilde{\sigma} + \tilde{\sigma}\tilde{W} \tag{12.58}$$

$\tilde{D}^e$ is the elastic part of the rate of deformation tensor $\tilde{D}$, λ and μ are the Lame' elastic constants, and $\phi(0 \leq \phi \leq 0.99)$ is the damage.

- Modules: material_libs/bammann_chiesa_johnson
  - bcjvpd_mig.h
  - bcjvpd_mig.C
  - bcjvpd.dat is the ASCII data file that lists the associated Fortran files.
- Physics: solid dynamics

Table 64: Input Parameters for `BAMMANN CHIESA JOHNSON`.

Parameter Name	Type	Description
`MATLABEL`	`string`	Material label in the `VP_data` file. If this parameter is provided, no other parameters are required. Enclose the string in single quotes.
`DATAFILE`	`string`	Name and location of the specially formatted `VP_data` file if different from the default `$ALEGRA_MIGDATA/VP_data`. Enclose the string in single quotes.
`RHO`	`real`	Initial density
`YM`	`real`	Young's modulus, used to determine the shear modulus.
`PR`	`real`	Poisson's ratio, used to determine the shear modulus.
`TEMP0`	`real`	Initial temperature
`HC`	`real`	Heat coefficient $(1/\rho C_v)$
`C1`	`real`	Coefficient and exponent for function $V(\theta)$
`C2`	`real`	Coefficient and exponent for function $V(\theta)$
`C3`	`real`	Parameter for function $Y(\theta)$
`C4`	`real`	Parameter for function $Y(\theta)$
`C5`	`real`	Coefficient and exponent for function $f(\theta)$
`C6`	`real`	Coefficient and exponent for function $f(\theta)$
`C7`	`real`	Coefficient and exponent for function $r_d(\theta)$
`C8`	`real`	Coefficient and exponent for function $r_d(\theta)$
`C9`	`real`	Parameter for function $h(\theta)$
`C10`	`real`	Parameter for function $h(\theta)$
`C11`	`real`	Parameter for function $r_s(\theta)$
`C12`	`real`	Parameter for function $r_s(\theta)$
`C13`	`real`	Coefficient and exponent for function $R_d(\theta)$
`C14`	`real`	Coefficient and exponent for function $R_d(\theta)$
`C15`	`real`	Parameter for function $H(\theta)$
`C16`	`real`	Parameter for function $H(\theta)$
`C17`	`real`	Coefficient and exponent for function $R_s(\theta)$
`C18`	`real`	Coefficient and exponent for function $R_s(\theta)$
`C19`	`real`	Parameter for function $Y(\theta)$

continued on next page

continued from previous page		
C20	real	Parameter for function $Y(\theta)$
A1	real	Initial value of backstress component α_{11}
A2	real	Initial value of backstress component α_{22}
A3	real	Initial value of backstress component α_{12}
A4	real	Initial value of backstress component α_{23}
A5	real	Initial value of backstress component α_{23}
A6	real	Initial value of scalar hardening variable κ
DEX	real	Value of exponent m in definition of damage variable ϕ
D0	real	Initial value of the damage variable ϕ
FS0	real	Initial value of the material spall strength p_0^f

Table 65: Registered Plot Variables for BAMMANN CHIESA JOHNSON.

Variable Name	Type	Mode	Description
DEFRATE	symtensor	INPUT	Rate of deformation tensor
TEMPERATURE	real	INPUT	Absolute temperature
PRESSURE	real	INPUT	Pressure
STRESS	real	INPUT	Cauchy stress
BACK_STRESS	real	INPUT	Back stress
EQPS	real	INPUT	Equivalent plastic strain
PLSNRT	real	OUTPUT	Plastic strain rate
DAMAGE	real	IOPUT	Damage fraction
KAPP	real	IOPUT	Scalar hardening variable k
BETA	real	IOPUT	Viscoplastic rate b
DAMR	real	IOPUT	Damage rate
BCJP	real	IOPUT	BCJ tensile pressure

```
$ Sample input for Bammann-Chiesa-Johnson
$ HY-80_STEEL (taken from cth VP_data file)

model 123 bammann chiesa johnson
   rho   = 7.831000e+00
   ym    = 2.069000e+12
   pr    = 3.000000e-01
   temp0 = 2.536947e-02
   hc    = 0.000000e+00
```

```
   c1    = 0.000000e+00
   c2    = 0.000000e+00
   c3    = 5.449000e+09
   c4    = 0.000000e+00
   c5    = 1.000000e+00
   c6    = 0.000000e+00
   c7    = 5.728000e-09
   c8    = 0.000000e+00
   c9    = 4.262000e+10
   c10   = 0.000000e+00
   c11   = 0.000000e+00
   c12   = 0.000000e+00
   c13   = 1.069000e-10
   c14   = 0.000000e+00
   c15   = 2.262000e+09
   c16   = 0.000000e+00
   c17   = 0.000000e+00
   c18   = 0.000000e+00
   a1    = 0.000000e+00
   a2    = 0.000000e+00
   a3    = 0.000000e+00
   a4    = 0.000000e+00
   a5    = 0.000000e+00
   a6    = 0.000000e+00
   dex   = 3.700000e+00
   d0    = 1.000000e-04
   fs0   = 3.670000e+10
   c19   = 0.000000e+00
   c20   = 0.000000e+00
end
```

12.5.5 Von Mises Yield

This model [9, 38] provides a calculation of yield stress based upon an isotropic hardening, von Mises criterion. When this model is sequenced with `LINEAR ELASTIC`, and `SIMPLE RADIAL RETURN`, it is exactly equivalent to `ELASTIC PLASTIC` for `BETA` = 1.0.

Generally, the above sequence of models is not usually used as the

ELASTIC PLASTIC model is more compact and efficient. However, if a non-linear elastic model were developed and needed to be used with a von Mises yield criterion, then that model sequenced with the VON MISES YIELD and SIMPLE RADIAL RETURN models would be appropriate.

- Modules: material_libs/standard_models
  - von_mises_yield.h
  - von_mises_yield.C
- Physics: solid dynamics

Table 66: Input Parameters for VON MISES YIELD.

Parameter Name	Type	Description
YOUNGS MODULUS	real	Young's Modulus of the material
POISSONS RATIO	real	Poisson's ratio of the material
YIELD STRESS0	real	Initial yield stress in uniaxial tension
HARDENING MODULUS	real	Hardening modulus for yield stress

Table 67: Registered Plot Variables for VON MISES YIELD.

Variable Name	Type	Mode	Description
STRESS	symtensor	INPUT	Trial Cauchy stress
YIELD_STRESS	real	IOPUT	Yield stress in uniaxial tension

```
material 10 copper
   model 11    $ linear elastic
   model 12    $ von mises yield
   model 13    $ simple radial return
   model 14    $ generic eos
   density     = 8.932
   temperature = 298.
end

model 11 linear elastic
   ...
```

```
end

model 12 von mises yield
   ...
end

model 13 simple radial return
end

model 14 generic eos
   ...
end
```

12.6 Plasticity Models

12.6.1 Simple Radial Return

This model [9, 38] performs stress relaxation to return the deviatoric stress onto the yield surface and computes the equivalent plastic strain incurred by this relaxation. This model is an extraction of the plasticity algorithm used in the `ELASTIC PLASTIC` model. This model has no input parameters. It provides a basic radial return algorithm to be used in conjunction with stress deviator and yield models. In particular, this model may be useful during development or prototyping of a new yield or stress deviator model.

- Modules: material_libs/standard_models
  - matmod_radret.h
  - matmod_radret.C
- Physics: solid dynamics

Table 68: Input Parameters for `SIMPLE RADIAL RETURN`.

Parameter Name	Type	Description
no parameters		

Table 69: Registered Plot Variables for `SIMPLE RADIAL RETURN`.

Variable Name	Type	Mode	Description
`STRESS`	`symtensor`	`IOPUT`	Cauchy stress tensor (`INPUT`: trial stress) (`OUTPUT`: stress on yield surface)
`EQPS`	`real`	`IOPUT`	Equivalent plastic strain

```
$ Example of request for simple radial return model.

model 43 simple radial return
end
```

12.6.2 EP Radial Return

This is the MIG implementation of the radial return plasticity algorithm used in CTH. This model performs stress relaxation to return the deviatoric stress onto the yield surface and computes the equivalent plastic strain incurred by this relaxation. This model is equivalent to the `SIMPLE RADIAL RETURN` model. However, it is implemented for use as a submodel in the `CTH ELASTIC PLASTIC` model. This model has no input parameters.

- Modules: material_libs/ep_radial_return
  - eprr_mig.h
  - eprr_mig.C
  - eprmig.F
  - ep_radial_return.dat is the associated data file
- Physics: solid dynamics

Table 70: Input Parameters for `EP RADIAL RETURN`.

Parameter Name	Type	Description
no parameters		

```
$ Example of request for ep radial return model.

model 43 ep radial return
end
```

12.7 Combined Models

12.7.1 CTH ELASTIC PLASTIC

This model replicates the elastic plastic algorithm in CTH [30]. It consists of three submodels, two of which may be user specified: an equation of state submodel and a yield stress submodel. The third model is the `EP RADIAL`

Table 71: Registered Plot Variables for EP RADIAL RETURN.

Variable Name	Type	Mode	Description
MATFRAC	real	INPUT	Material fraction of element volume
DEFRATE	symtensor	INPUT	Rate of deformation tensor
SPIN	antitensor	INPUT	Spin tensor, antisymmetric part of velocity gradient
SHEAR_MODULUS	real	INPUT	Shear modulus
YIELD_STRESS	real	INPUT	Yield stress in uniaxial tension
STRESS	symtensor	IOPUT	Cauchy stress tensor (INPUT: trial stress) (OUTPUT: stress on yield surface)
EQPS	real	IOPUT	Equivalent plastic strain

RETURN model. The calculational sequence of this model begins with the equation of state model to compute pressure, temperature, and sound speed. Next, the shear modulus is calculated as a function of bulk sound speed and constant Poisson's ratio.

$$G = \frac{3(1-2\nu)}{1(1+\nu)}\rho C^2 \tag{12.59}$$

The shear modulus is used to compute an elastic deviatoric stress increment. Next, the yield model is applied to determine if the deviatoric stress increment exceeds the current yield criterion. Then, the EP RADIAL RETURN model is used to perform the plasticity calculation to enforce the yield criterion.

This model is an example of a combined model which has a specific sequence and set of submodels to be used in computing the material state. Currently, the Steinberg-Guinan-Lund, Johnson-Cook EP, and Zerilli-Armstrong models are compatible with CTH ELASTIC PLASTIC and may be used as submodels. A sample input fragment is shown below.

- Modules: material_libs/combined_models
  - cth_ep.h
  - cth_ep.C

Table 72: Compatible Models for CTH ELASTIC PLASTIC.

Submodel Type	Models
Equation of State Models	KEOS MieGruneisen KEOS Sesame MG US UP KEOS Sesame
Yield Models	STEINBERG GUINAN LUND JOHNSON COOK EP ZERILLI ARMSTRONG BAMMANN CHIESA JOHNSON

- Physics: solid dynamics

Table 73: Input Parameters for CTH ELASTIC PLASTIC.

Parameter Name	Type	Description
EOS MODEL	int	Model id of equation of state model
YIELD MODEL	int	Model id of yield stress model
POISSONS RATIO	real	Poisson's ratio

Table 74: Registered Plot Variables for CTH ELASTIC PLASTIC.

Variable Name	Type	Mode	Description
DENSITY	real	INPUT	Material density
SOUND_SPEED	real	IOPUT	Sound speed
SHEAR_MODULUS	real	OUTPUT	Shear modulus

```
material 10 'OFHC Copper'
   model 11    $ cth ep
   density     = 8.932
   temperature = 298.
end

model 11 cth ep
   eos model     = 12   $ mg us up
```

```
   yield model   = 13   $ johnson cook
   poisson ratio = 0.3
end

model 12 mg us up
   ...
end

model 13 johnson cook ep
   ...
end
```

12.7.2 BFK CONCRETE

This model replicates the brittle fracture kinetics (BFK) concrete model in CTH [31]. This combined model consists of submodels which may NOT be user specified, including models for the equation of state, yield, radial return, and fracture. The calculational sequence of this model begins with a call to the equation of state to compute pressure, temperature, and sound speed. This is followed by the calculation of the flow stress. The radial return algorithm is called to enforce the yield criterion and provide the plastic strain rate. Then follows the calculation of the extra variables, including the equivalent plastic strain. Finally, the fracture algorithm is called to compute the void insertion required based on the fracture pressure calculated previously. In Eulerian problems, after the remap step the equation of state and the fracture algorithms are called again.

Currently only one model is predefined for concrete. This model, `SAC5`, represents small aggregate concrete with an unconfined compressive strength around 6,000 psi. The input parameter `COSFAC` can be used to scale all strength parameters. For example, if the desired material has an unconfined compressive strength of 5,000 psi, the default parameters for `SAC5` can be used with `COSFAC` = 0.8333.

- Modules: material_libs/bfk_concrete
  - bfk_concrete_mig.h
  - bfk_concrete_mig.C

- MIG driver routines:
  - elvpco.F
  - elivco.F
  - eoscos.F
  - eoscov.F
  - eoscox.F
- MIG input, data check, and extra variable routines:
  - eoscoi.F
  - concck.F
  - concxv.F
- MIG utility routines:
  - random.F
  - splint.F
  - gauss.F
  - splco.F
- MIG routines, miscellaneous:
  - condis.F
  - conmfs.F
  - conhro.F
- Physics: solid dynamics

Table 75: Input Parameters for `BFK CONCRETE`.

Parameter Name	Type	Description
continued on next page		

continued from previous page

MATLABEL	string	Material label in the EOS_data file. If this parameter is provided, no other parameters are required. Enclose the string in single quotes. The only material currently predefined in the EOS_data file is called SAC5.
DATAFILE	string	Name and location of the specially formatted EOS_data file if different from the default $ALEGRA_MIGDATA/EOS_data. Enclose the string in single quotes.
COCRX	real	Switch for crack visualization (not used).
COSC0	real	Unconfined compressive strength, s_{c0}.
COST0	real	Unconfined tensile strength, s_{t0}.
COSD0	real	Brittle-ductile transition stress, s_{d0}.
COSHRI	real	Instantaneous shear modulus, μ_i.
COCC	real	Compression at crush, μ_c
COFRA1	real	Fragment size coefficient, A_1
COSHRF	real	Failed shear modulus, μ_f.
COSTI	real	Instantaneous unconfined tensile strength, s_{ti}.
COCH	real	Hardening coefficient, C_h
COK0	real	Initial bulk modulus, k_0
COKN	real	Limiting bulk modulus, k_0
COCV	real	Specific heat, C_v
CORH0	real	Reference density, ρ_0.
COY0	real	Initial yield stress, Y_0.
COREC	real	Recovery compression, μ_{rec}.
COTAU	real	Fundamental time to failure, τ_0
COSCI	real	Instantaneous unconfined compressive strength, s_{ci}.
COGRU	real	Grüneisen coefficient, Γ_0
COYMIN	real	Residual flow stress, Y^0_{res}
COYMAX	real	Limiting flow stress, Y_{max}.
COKF	real	Failed bulk modulus, k_0
COCRXS	real	Critical overload for crack growth (not used)
COCRXE	real	Critical strain for crack growth, ϵ_{grow} (not used)
COCRXV	real	Crack growth velocity (not used)

continued on next page

continued from previous page

COCRXM	real	Maximum number of crack tracer particles per cell (not used).
COQDF	real	Maximum fragment size for CDF, λ_{max}. If 0, no CDF made.
COFRA2	real	Fragment size coefficient, A_2
COTSOF	real	Thermal softening temperature, T_{soft}
COED1	real	Equivalent plastic strain at onset of ductile damage, ϵ_{d1}
COED2	real	Equivalent plastic strain at full ductile damage, ϵ_{d1}
COPD1	real	Pressure at onset of compressive damage, p_{d1}
COPD2	real	Pressure at full compressive damage, p_{d2}
COTSPL	real	Principal stress at spall, T_{spall}
COBFIC	real	Critical brittle fracture impulse, Ω_1
CODIL	real	Dilantancy parameter (not currently used).
COTTRJ	real	Trajectory time parameter.
COFTRJ	real	Trajectory parameter.
COFMOB	real	Surface mobility coefficient, C_{surf}
COFDEP	real	Surface effect depth, D_{surf}
COSFAC	real	Multiplier for strength parameters.
COTREL	real	Time at which discard flag is set.
COVREL	real	Minimum velocity for discard.
COBREL	real	Minimum value of COBFIC (Ω_1) for discard.
COTDIS	real	Discard time (greater than or equal to COTREL)
NXSRF [NYSRF] [NZSRF]	int	Number of potential spall surfaces in planes aligned with X, Y, and Z coordinated planes.
CXSRF0 CXSRF1 CXSRF2 ... CXSRF9	real	Potential spall surfaces defined by a constant value of x-coordinate. (This basic capability was brought from CTH, later to be extended so that the code can find the surface.)
CYSRF0 CYSRF1 CYSRF2 ... CYSRF9	real	Potential spall surfaces defined by a constant value of y-coordinate. (This basic capability was brought from CTH, later to be extended so that the code can find the surface.)

continued on next page

continued from previous page		
CZSRF0 CZSRF1 CZSRF2 ... CZSRF9	real	Potential spall surfaces defined by a constant value of z-coordinate. (This basic capability was brought from CTH, later to be extended so that the code can find the surface.)
NTBLCO	int	Number of data pairs for experimental Hugoniot data (data given by CTBLXn and CTBLYn.) These data are used for a cubit spline fit for the loading part of the curve. At compressions greater than those provided in the table, the reference curve follows a straight line in $\rho - \mu$ space with slope COKN.
CTBLX1 CTBLX2 CTBLX3 ... CTBLXF	real	Data for compression, $\mu = (\rho/\rho_0) - 1$, used for the experimental Hugoniot data.
CTBLY1 CTBLY2 CTBLY3 ... CTBLYF	real	Pressure data for the experimental Hugoniot data.

```
material 10 concrete
   model 11   $ bfk concrete
end

model 11 bfk concrete
   matlable = 'SAC5'     $ predefined concrete model
   cotau    = 1000.e-6   $ modify parameter
end
```

12.8 Fracture Models

12.8.1 Pressure Dependent Fracture

This pressure dependent fracture model [24] uses a void insertion algorithm to allow the volume occupied by a material in an element to decrease, thus

Table 76: Registered Plot Variables for BFK CONCRETE.

Variable Name	Type	Mode	Description
DENSITY	real	IOPUT	Material density
ENERGY	real	INPUT	Specific energy.
PRESSURE	real	OUTPUT	Pressure.
TEMPERATURE	real	OUTPUT	Temperature
SOUND_SPEED	real	OUTPUT	Sound speed
DPDRHO	real	OUTPUT	Derivative $dP/d\rho$.
STRESS	symtensor	IOPUT	Cauchy stress tensor.
YIELD_STRESS	real	IOPUT	Yield in tension.
SHEAR_MODULUS	real	OUTPUT	Shear modulus
DEFRATE	symtensor	INPUT	Rate of deformation tensor.
PLAS_STRN_RATE	real	OUTPUT	Scalar plastic strain rate.
SPIN	antitensor	INPUT	Spin tensor, antisymmetric part of velocity gradient
EQPS	real	OUTPUT	Equivalent plastic strain.
MD	real	IOPUT	Maximum density, $1/\bar{\nu}$.
DMG	real	IOPUT	Damage variable, ϕ.
FS	real	IOPUT	Fracture stress, $-T_b$.
OL	real	IOPUT	Overload, Ω.
BFI	real	IOPUT	Brittle fracture impulse, Ω.
STN	real	IOPUT	Strength, Y_{inf}
MFS	real	IOPUT	Mean fragment size, λ_f.
REL	real	IOPUT	Release (discard) flag (not yet implemented.)

allowing the density to increase and the pressure to relax to zero. This algorithm is triggered when the pressure in the material is less than the fracture pressure. A Newton iteration scheme is used in which the density is increased and the equation of state model computes the new pressure. When the pressure converges to the fracture pressure, the volume of void inserted is determined by the density change required to produce the fracture pressure. In subsequent cycles, the fracture pressure is gradually decreased until it is zero. Note that the material input must have an equation of state model preceding this fracture model to compute pressure and energy. A typical application of this model would use the following input.

- Modules: material_libs/standard_models
  - frac_presdep.h
  - frac_presdep.C
- Physics: hydrodynamics

Table 77: Input Parameters for `FRAC PRESDEP`.

Parameter Name	Type	Description
`INIT FRAC PRES`	`real`	Initial fracture pressure
`DENSITY TOLERANCE`	`real`	Density tolerance for convergence of iteration
`PRESSURE TOLERANCE`	`real`	Pressure tolerance for convergence of iteration
`MAX NUM OF ITERATIONS`	`real`	Maximum number of iterations per cycle
`CYCLES TO FAIL`	`real`	Number of cycles to relax from the initial fracture pressure to zero
`FAILURE INCREMENT`	`real`	

Table 78: Registered Plot Variables for `FRAC PRESDEP`.

Variable Name	Type	Mode	Description
`FAILURE FRACTION`	`real`	`IOPUT`	
`FRACTURE PRESSURE`	`real`	`IOPUT`	Current fracture pressure
`DENSITY`	`real`	`IOPUT`	Material density
`ENERGY`	`real`	`IOPUT`	Specific internal energy
`PRESSURE`	`real`	`IOPUT`	Pressure

```
material 10 'some material'
   model 11  $ mg us up
   model 12  $ frac pres dep
   ...
end

model 11 mg us up
```

```
   ...
end

model 12 frac presdep
   init frac pres     = -5.e10
   density tolerance  = 1.e-6
   pressure tolerance = 1.e+2
end
```

12.9 KEOS Reactive Burn Models

Five Kerley reactive burn models [21, 22] are available in ALEGRA.

- KEOS ARB
- KEOS FFRB
- KEOS HVRB
- KEOS IGRB
- KEOS Ptran

For each of these KEOS models, the material decomposition is included in the equation of state, and the time evolution of the reaction is described by a rate equation. Detailed descriptions of these models and tips for using the reactive burn models are provided in these references. The basic equations will be presented here as a summary, followed by the specific rate equation and input parameters for each burn model.

The reactive burn models require the equation of state for the unreacted material, the reaction products, and the partially reacted explosive. They are called "composite" models because they are constructed from the basic Kerley EOS models:

- KEOS Sesame
- KEOS MieGruneisen
- KEOS JWL

- KEOS Ideal Gas

for the unreacted materials and reaction products. The partially reacted explosive is described by the following equations.

$$P(\rho, T, \lambda) = (1 - \lambda)P_{UR}(\rho, T) + \lambda P_{RP}(\rho, T) \tag{12.60}$$

$$E(\rho, T, \lambda) = (1 - \lambda)E_{UR}(\rho, T) + \lambda E_{RP}(\rho, T) \tag{12.61}$$

where λ is the extent of reaction ($\lambda = 0.0$ for no reaction, $\lambda = 1.0$ for complete reaction). The subscripts UR and RP denote the equation of state for the unreacted explosive and the reaction products, respectively. The equation for advancing λ in time is given for each of the five models in the following subsections.

- Modules: material_libs/kerley_eos
  - rbn_mig.h
  - rbn_mig.C
- Physics: hydrodynamics

The extra HISPLT variables for the KEOS reactive burn models are listed in Table 79. Other HISPLT variables are listed in Section 13.3 on page 298.

Table 79: Extra HISPLT Variables for KEOS Reactive Burn Models.

Variable Name	Description
EXT_REACTION	The extent of reaction of the burn. The value ranges from 0 to 1.
ALPI	The porosity parameter if the initial state submodel is a porosity model. This parameter represents the ratio solid density/porous density.
ALPF	The porosity parameter if the final state submodel is a porosity model. This parameter represents the ratio solid density/porous density.

Table 80: Registered Plot Variables of KEOS Reactive Burn Models.

Variable Name	Type	Mode	Description
DENSITY	real	INPUT	Material density
ENERGY	real	INPUT	Specific internal energy per unit mass
PRESSURE	real	OUTPUT	Pressure
TEMPERATURE	real	OUTPUT	Absolute temperature
SOUND SPEED	real	OUTPUT	Bulk sound speed
DPDRHO	real	OUTPUT	Derivative of pressure with respect to density
EXT_REACTION	real	OUTPUT	Extent of reaction parameter, non dimensional.
ALPI	real	OUTPUT	Porosity parameter, alpha, from submodel for initial, unreacted state.
ALPF	real	OUTPUT	Porosity parameter, alpha, from submodel for final state for reaction products.

12.9.1 KEOS ARB

The KEOS ARB (Arrhenius Reactive Burn) model can be used for the initiation and propagation of detonations in homogeneous explosives. The rate law is a function of the temperature.

$$\frac{d\lambda}{dt} = (1-\lambda)F\exp\left(\frac{\Theta}{T}\right) \tag{12.62}$$

where F is the frequency factor, and Θ is the activation temperature.

$$\Theta = \Theta_0(1 + A_P P) \tag{12.63}$$

The parameters F, Θ_0, and A_P are obtained by fitting data from wedge tests and other experiments.

- Modules: material_libs/kerley_eos
  - arb_mig.h

– arb_mig.C

– arb.doc is the ASCII data file that lists associated Fortran routines.

- Physics: hydrodynamics

Table 81: Input Parameters for `KEOS ARB`.

Parameter Name	Type	Description
`MATLABEL`	`string`	Label listed in the `EOS_data` file for a predefined material for the `KEOS ARB` model. If this keyword is used, no other keyword is required. Enclose the `string` in single quotes.
`DATAFILE`	`string`	Name and location of the specially formatted `EOS_data` file if different than the default `$ALEGRA_MIGDATA/EOS_data`. Enclose the `string` in single quotes.
`AT`	`real` (0.)	Rate equation constant Θ_0. Required unless `MATLABEL` given.
`FF`	`real` (1.e10 s^{-1})	Rate equation constant F. Required unless `MATLABEL` given.
`ATP`	`real`	Rate equation constant A_P. Required unless `MATLABEL` given.
`TI`	`real`	Threshold temperature [initial temp].
`RMIN`[a]	`real` (0.)	Minimum density, unreacted explosive. Reaction is complete for densities < `RMIN`.
`RMAX`[a]	`real` (1.e30)	Maximum density, unreacted explosive. Reaction is complete for densities > `RMAX`.
`TMAX`[a]	`real` (1.e30)	Maximum temperature, unreacted explosive. Reaction is complete for temperatures > `TMAX`.
`VF`[a]	`real` (1.)	Volume ratio, initial state/final state
`TF`[a]	`real` (1.)	Temperature ratio, initial state/final state
`ESFT`[a]	`real`	Shift in energy zero (optional).
`EOSUR MODNUMBER`	`int`	The model number in the input deck that will be used for the unreacted material. [Required unless defined in `EOS_data` file or by `EOSUR MATLABEL`.]

continued on next page

continued from previous page		
EOSRP MODNUMBER	int	The model number in the input deck that will be used for the reacted products. [Required unless defined in EOS_data file or by EOSRP MATLABEL.]
EOSUR MODLABEL	string	Model label listed in the EOS_data file for a predefined material for the unreacted material. Enclose the string in single quotes. If this parameter in used, EOSUR MATLABEL is also required.
EOSRP MODLABEL	string	Model label listed in the EOS_data file for a predefined material for the reacted products. Enclose the string in single quotes. If this parameter in used, EOSRP MATLABEL is also required.
EOSUR MATLABEL	string	Label listed in the EOS_data file for a predefined material for the unreacted material. Enclose the string in single quotes. If this parameter in used, EOSUR MODLABEL is also required.
EOSRP MATLABEL	string	Label listed in the EOS_data file for a predefined material for the reacted products. Enclose the string in single quotes. If this parameter in used, EOSRP MODLABEL is also required.
EOSUR	string	This keyword is only read from the EOS_data file. It points to the model for the unreacted material in the EOS_data file.
EOSRP	string	This keyword is only read from the EOS_data file. It points to the model for the reacted products in the EOS_data file.
[a] This parameter is normally not input by the user, being left to its default value.		

Two sample inputs for KEOS ARB model, in cgsK units, are:

```
$ Sample 1:
$ Initial (UR) and final (RP) states are specified in
$ EOS_data for PETN.

model 201 keos arb
   matlabel = 'PETN'
end
```

```
$ Sample 2:
$ The user specifies initial (UR) and final (RP) states.

model 22 keos arb
   eosur modnumber = 221  $ UR is model 221 below.
   eosrp modnumber = 222  $ RP is model 222 below.
end

model 221 keos miegrun
   matlabel = 'petn'
   rp       = 1.75
end

model 222 keos jwl
   matlabel = 'petn'
   brn      = 0.      $ EOS_data file assumes jwl is
end                   $ programmed burn.
```

12.9.2 KEOS FFRB

The `KEOS FFRB` (Forest Fire Reactive Burn) model can be used for the initiation and propagation of detonations in heterogeneous explosives. The rate law is pressure dependent.

$$\frac{d\lambda}{dt} = (1-\lambda)^f R(P) \tag{12.64}$$

where f is the order of the reaction ($0 \leq f \leq 1$). The function $R(P)$ is derived from the Pop plot, using the "single-curve buildup principle" and fit to an analytic expression. The required input parameters are the Pop plot variables and P_{max}.

- Modules: material_libs/kerley_eos
  - ffrb_mig.h
  - ffrb_mig.C
  - ffrb.doc is the ASCII data file that lists the associated Fortran routines.

- Physics: hydrodynamics

Table 82: Input Parameters for KEOS FFRB.

Parameter Name	Type	Description
MATLABEL	string	Label listed in the EOS_data file for a predefined material for the KEOS FFRB model. If this keyword is used, no other keyword is required. Enclose the string in single quotes.
DATAFILE	string	Name and location of the specially formatted EOS_data file if different than the default $ALEGRA_MIGDATA/EOS_data. Enclose the string in quotes.
P1 X1	real	Pressure and run distance at 1st point on Pop plot. Required unless MATLABEL given.
P2 X2	real	Pressure and run distance at 2nd point on Pop plot. Required unless MATLABEL given.
PMAX	real	Pressure for instantaneous reaction. Required unless MATLABEL given.
PMIN	real (0.01*PMAX)	Threshold pressure for reaction.
FR	real (0.)	Order of reaction f.
NRH	real (0.)	Hugoniot option. NRH = 0 for reactive Hugoniot, NRH = 1 for nonreactive Hugoniot.
RMIN	real (0.)	Minimum density, unreacted explosive. Reaction is complete for densities < RMIN.
RMAX[a]	real (1.e30)	Maximum density, unreacted explosive. Reaction is complete for densities > RMAX.
TMAX[a]	real (1.e30)	Maximum temperature, unreacted explosive. Reaction is complete for temperatures > TMAX.
VF[a]	real (1.)	Volume ratio, initial state/final state.
TF[a]	real (1.)	Temperature ratio, initial state/final state.
ESFT[a]	real	Shift in energy zero (optional)

continued on next page

continued from previous page		
EOSUR MODNUMBER	int	The model number in the input deck that will be used for the unreacted material. [Required unless defined in EOS_data file or by EOSUR MATLABEL.]
EOSRP MODNUMBER	int	The model number in the input deck that will be used for the reacted products. [Required unless defined in EOS_data file or by EOSRP MATLABEL.]
EOSUR MODLABEL	string	Model label listed in the EOS_data file for a predefined material for the unreacted material. Enclose the string in single quotes. If this parameter in used, EOSUR MATLABEL is also required.
EOSRP MODLABEL	string	Model label listed in the EOS_data file for a predefined material for the unreacted material. Enclose the string in single quotes. If this parameter in used, EOSRP MATLABEL is also required.
EOSUR MATLABEL	string	Label listed in the EOS_data file for a predefined material for the unreacted material. Enclose the string in single quotes. If this parameter in used, EOSUR MODLABEL is also required.
EOSRP MATLABEL	string	Label listed in the EOS_data file for a predefined material for the reacted products. Enclose the string in single quotes. If this parameter in used, EOSRP MODLABEL is also required.
EOSUR	string	This keyword is only read from the EOS_data file. It points to the model for the unreacted material in the EOS_data file.
EOSRP	string	This keyword is only read from the EOS_data file. It points to the model for the reacted products in the EOS_data file.
[a] This parameter is normally not input by the user, being left to its default value.		

```
$ Sample input for ffrb model, in cgsK units.
$ Note that initial temperature and density are from the
```

```
$ initial state given eosur model.

model 221 keos ffrb
   eosur modnumber = 222
   eosrp modnumber = 223
   x1   = 0.05       $ cm
   p1   = 18.1567e10 $ dyn/cm^2
   x2   = 0.50       $ cm
   p2   = 4.4315e10  $ dyn/cm^2
   fr   = 1.0        $
   pmin = 1.e9       $ dyn/cm^2
   pmax = 3.53e11    $ dyn/cm^2
   nrh  = 1.         $
end

model 222, keos miegrun
   matlabel = 'pbx9404'
end

model 223, keos sesame
   matlabel = 'pbx9404_dp'
end
```

12.9.3 KEOS HVRB

The `KEOS HVRB` (History Variable Reactive Burn) model can be used for the initiation and propagation of detonations in heterogeneous explosives. The rate law is pressure dependent with a time delay before initiation of rapid reaction. The equation is written in integral form by introducing a history variable.

$$\lambda = 1 - \left(1 - \frac{\phi^M}{X}\right)^X \tag{12.65}$$

$$\phi(t) = \frac{1}{\tau_0}\int_0^t \left(\frac{P - P_I}{P_R}\right)^z dt \tag{12.66}$$

The parameters P_R and z determine the pressure dependence of the time

and distance to detonation (these are usually fit to wedge test data). The exponent M controls the time delay to pressure buildup behind the shock front, and X determines the rate at which the reaction goes to completion. P_I is the threshold pressure for initiation, and the coefficient τ_0 is used to make ϕ dimensionless and is not an independent constant.

- Modules: material_libs/kerley_eos
  - hvrb_mig.h
  - hvrb_mig.C
  - hvrb.doc is the ASCII data file that lists associated Fortran routines.
- Physics: hydrodynamics

Table 83: Input Parameters for `KEOS HVRB`.

Parameter Name	Type	Description
`MATLABEL`	`string`	Label listed in the `EOS_data` file for a predefined material for the `KEOS HVRB` model. If this keyword is used, no other keyword is required. Enclose the `string` in single quotes.
`DATAFILE`	`string`	Name and location of the specially formatted `EOS_data` file if different than the default `$ALEGRA_MIGDATA/EOS_data`.
`PR`	`real`	Rate equation constant P_R. This parameter, along with `ZR`, determines the overall rate of the reaction. It is closely related to constants in Pop plot expressions. Required unless `MATLABEL` given.
`ZR`	`real`	Rate equation constant z. This parameter, along with `PR`, determines the overall rate of the reaction. It is closely related to constants in Pop plot expressions. Required unless `MATLABEL` given.

continued on next page

continued from previous page

MR	real	Rate equation constant M. This parameter primarily affects the shape of the pressure wave behind the shock wave during buildup to detonation. (MR typically ranges from about 1 to 2.) Required unless MATLABEL given.
PI	real	Threshold pressure for shock initiation, P_I. Required unless MATLABEL given.
XR	real	Rate equation constant. Required unless MATLABEL given.
RMIN[a]	real (0.)	Minimum density, unreacted explosive. Reaction is complete for densities < RMIN.
RMAX[a]	real (1.e30)	Maximum density, unreacted explosive. Reaction is complete for densities > RMAX.
TMAX[a]	real (1.e30)	Maximum temperature, unreacted explosive. Reaction is complete for temperatures > TMAX.
VF[a]	real (1.)	Volume ratio, initial state/final state
TF[a]	real (1.)	Temperature ratio, initial state/final state
ESFT[a]	real	Shift in energy zero (optional).
EOSUR MODNUMBER	int	The model number in the input deck that will be used for the unreacted material. [Required unless defined in EOS_data file or by EOSUR MATLABEL.]
EOSRP MODNUMBER	int	The model number in the input deck that will be used for the reacted products. [Required unless defined in EOS_data file or by EOSRP MATLABEL.]
EOSUR MODLABEL	string	Model label listed in the EOS_data file for a predefined material for the unreacted material. Enclose the string in single quotes. If this parameter in used, EOSUR MATLABEL is also required.

continued on next page

continued from previous page		
EOSRP MODLABEL	string	Model label listed in the EOS_data file for a predefined material for the unreacted material. Enclose the string in single quotes. If this parameter in used, EOSRP MATLABEL is also required.
EOSUR MATLABEL	string	Label listed in the EOS_data file for a predefined material for the unreacted material.
EOSRP MATLABEL	string	Label listed in the EOS_data file for a predefined material for the reacted products.
EOSUR	string	This keyword is only read from the EOS_data file. It points to the model for the unreacted material in the EOS_data file.
EOSRP	string	This keyword is only read from the EOS_data file. It points to the model for the reacted products in the EOS_data file.
[a] This parameter is normally not input by the user, being left to its default value.		

```
$ Simple sample input for keos hvrb. Alternative, the user can
$ specify models for the unreacted initial state and the final
$ reacted products as shown in keos arb or keos ffrb.
$ The user can also modify parameters given in the EOS_data
$ file as shown below.

model 161 keos hvrb
   matlabel = 'HMX'
   rmin = 0.2
end
```

12.9.4 KEOS IGRB

The KEOS IGRB (Ignition and Growth Reactive Burn) model describes the initiation and propagation of detonations in heterogeneous explosives. It is a three-step model that describes the initiation of reaction in hot spots, followed by propagation of reaction to the rest of the explosive, and a rapid completion phase that occurs during coalescence of the hot spot regions. The

rate law is a function of pressure and density in the following equation.

$$\lambda = (1-\lambda)^{s_0} G_0 \frac{\rho}{(\rho_0 - 1 - a_0)^{y_0}} + (1-\lambda)^{s_1} \lambda^{q_1} G_1 P + (1-\lambda)^{s_2} \lambda^{q_2} G_2 P^{y_2} \quad (12.67)$$

The 12 constants in the above equation are chosen to fit experimental data for a given explosive. Three other constants are also used in the model. `W0` is the value of λ at which the first term (ignition) is turned *off*, `W1` is the value of λ at which the second term (growth) is turned *off*, and `W2` is the value of λ at which the third term (completion) is turned *on*.

- Modules: material_libs/kerley_eos
  - igrb_mig.h
  - igrb_mig.C
  - igrb.doc is the ASCII data file that lists associated Fortran routines.
- Physics: hydrodynamics

Table 84: Input Parameters for `KEOS IGRB`.

Parameter Name	Type	Description
`MATLABEL`	`string`	Label listed in the `EOS_data` file for a predefined material for the `KEOS IGRB` model. If this keyword is used, no other keyword is required. Enclose the `string` in single quotes.
`DATAFILE`	`string`	Name and location of the specially formatted `EOS_data` file if different than the default `$ALEGRA_MIGDATA/EOS_data`. Enclose the `string` in single quotes.
`G0` (or `GO`)	`real`	Ignition parameter G_0. Required unless `MATLABEL` given.
`S0` (or `SO`)	`real` (0.667)	Ignition parameter s_0.
`A0` (or `AO`)	`real` (0.0)	Ignition parameter a_0.

continued on next page

continued from previous page

Y0 (or YO)	real (20.0)	Ignition parameter y_0.
W0	real (0.3)	Ignition turnoff parameter.
G1	real	Growth parameter G_1. Required unless MATLABEL given.
S1	real (0.667)	Growth parameter s_1.
Q1	real (0.111)	Growth parameter q_1.
Y1	real (1.0)	Growth parameter y_1.
W1	real (0.5)	Growth turnoff parameter.
G2	real	Completion parameter G_2. Required unless MATLABEL given.
S2	real (0.333)	Completion parameter s_2.
Q2	real (1.0)	Completion parameter q_2.
Y2	real (2.0)	Completion parameter y_2.
W2	real (0.0)	Completion turn-on parameter.
NSUB	real (1)	Number of subcycles in one time step
RMIN[a]	real (0.0)	Minimum density, unreacted explosive. Reaction is complete for densities < RMIN.
RMAX[a]	real (1.e30)	Maximum density, unreacted explosive. Reaction is complete for densities > RMAX.
TMAX[a]	real (1.e30)	Maximum temperature, unreacted explosive. Reaction is complete for temperatures > TMAX.
VF[a]	real (1.0)	Volume ratio, initial state/final state
TF[a]	real (1.0)	Temperature ratio, initial state/final state
ESFT[a]	real	Shift in energy zero (optional)
EOSUR MODNUMBER	int	The model number in the input deck that will be used for the unreacted material. [Required unless defined in EOS_data file or by EOSUR MATLABEL.]
EOSRP MODNUMBER	int	The model number in the input deck that will be used for the reacted products. [Required unless defined in EOS_data file or by EOSRP MATLABEL.]

continued on next page

continued from previous page		
EOSUR MODLABEL	string	Model label listed in the EOS_data file for a predefined material for the unreacted material. Enclose the string in single quotes. If this parameter in used, EOSUR MATLABEL is also required.
EOSRP MODLABEL	string	Model label listed in the EOS_data file for a predefined material for the unreacted material. Enclose the string in single quotes. If this parameter in used, EOSRP MATLABEL is also required.
EOSUR MATLABEL	string	Label listed in the EOS_data file for a predefined material for the unreacted material.
EOSRP MATLABEL	string	Label listed in the EOS_data file for a predefined material for the reacted products.
EOSUR	string	This keyword is only read from the EOS_data file. It points to the model for the unreacted material in the EOS_data file.
EOSRP	string	This keyword is only read from the EOS_data file. It points to the model for the reacted products in the EOS_data file.
[a] This parameter is normally not input by the user, being left to its default value.		

```
$ Sample input for keos igrb model. As with other reactive
$ burn models, the user can specify individual models for
$ the initial and final states, and model parameters
$ specified in the EOS_data file can be changed. The
$ EOS_data file location and name can also be modified from
$ the default value.

model 211 keos igrb
   datafile = '/home/svpetne/mynewdata/EOS_data.new'
   matlabel = 'PETN'
end
```

12.9.5 KEOS Ptran

The `KEOS Ptran` (Phase Transition Reactive Burn) model [23] describes a material that has a transition between two phases. This model is similar to the two-state reactive burn models, but the transition is described by a phase boundary rather than a rate equation. The phase boundary is input by the user.

The user specifies the pressure in the transition region by the formula:

$$P(\rho, T, \lambda) = P_T + \beta_T \left(1 - \frac{\rho_T}{\rho}\right) + A_T(T - T_0) + A_\lambda \lambda \tag{12.68}$$

where P_T and ρ_T are the transition pressure and density of phase 1 at room temperature T_0, β_T is the bulk modulus in the transition region, and A_T and A_λ are derivatives of the transition pressure with respect to T and λ, respectively. P_T, β_T, A_T and A_λ are input parameters. λ is the mass fraction of phase 2.

- Modules: material_libs/kerley_eos
  - ptran_mig.h
  - ptran_mig.C
  - ptran.doc is the ASCII data file that lists associated Fortran routines.
- Physics: hydrodynamics

Table 85: Input Parameters for `KEOS Ptran`.

Parameter Name	Type	Description
`MATLABEL`	`string`	Label listed in the `EOS_data` file for a predefined material for the `KEOS Ptran` model. If this keyword is used, no other keyword is required. Enclose the `string` in single quotes.

continued on next page

continued from previous page

DATAFILE	string	Name and location of the specially formatted EOS_data file if different than the default $ALEGRA_MIGDATA/EOS_data. Enclose the string in single quotes.
PT	real	Pressure at start of transition.
BT	real	Bulk modulus in transition region.
AT	real	Temperature derivative of transition pressure.
AX	real	Spatial derivative of transition pressure.
HF	real (0.0)	Hysteresis flag. HF = 0.0 for reversible case and HF = 1.0 for irreversible case
RMIN[a]	real (0.0)	Minimum density for initial state
RMAX[a]	real (1.e30)	Maximum density for initial state
TMAX[a]	real (1.e30)	Maximum temperature for initial state
VF[a]	real (1.0)	Volume ratio, initial state/final state
TF[a]	real (1.0)	Temperature ratio, initial state/final state
ESFT[a]	real	Shift in energy zero (optional)
EOSUR MODNUMBER	int	The model number in the input deck that will be used for the initial phase of the material. [Required unless defined in EOS_data file or by EOSUR MATLABEL and EOSUR MODLABEL.]
EOSRP MODNUMBER	int	The model number in the input deck that will be used for the final phase of the material. [Required unless defined in EOS_data file or by EOSRP MATLABEL and EOSRP MODLABEL.]
EOSUR MODLABEL	int	The model type that will be used for the initial phase of the material. [If this parameter is input, EOSUR MATLABEL is also required.]
EOSRP MODLABEL	int	The model type that will be used for the final phase of the material. [If this parameter is input, EOSUR MATLABEL is also required.]
EOSUR MATLABEL	int	The model type that will be used for the initial phase of the material. [If this parameter is input, EOSUR MODLABEL is also required.]

continued on next page

continued from previous page		
EOSRP MATLABEL	int	The model type that will be used for the final phase of the material. [If this parameter is input, EOSUR MODLABEL is also required.]
[a] This parameter is normally not input by the user, being left to its default value.		

```
$ Iron, PTRAN model, phase 1--alpha, phase 2--epsilon.
$ Note esft for epsilon used to include transition energy.
$ Set BT=1.85e12 to match transition pressure to Rayleigh line.
$ Set HF=1 to make transition irreversible.

model 231 keos ptran
   eosur modnumber = 232
   eosrp modnumber = 233
   pt = 13.0e10
   bt =  1.0e9
   at = -1.034082e7  $ -1.2e11/11604.5
   ax = -4.0e10
end

model 232 keos miegrun
   r0 = 7.87
   t0 = 298.
   cs = 4.6e5
   s1 = 1.46
   g0 = 1.7
   cv = 4.601663e6   $ 5.34e10/11604.5
end

model 233 keos miegrun
   r0 = 8.29
   t0 = 298.
   cs = 4.6e5
   s1 = 1.51
   g0 = 2.4
   cv = 4.515490e+6  $ 5.24e10/11604.5
end
```

12.10 Burn Models

12.10.1 Programmed Burn JWL

```
MODEL model_id_number PROGRAMMED BURN JWL
   [parameter = value]
   ...
END
```

This model is nearly identical to the `JWL` equation of state model [21] (see Section 12.3.3 on page 212). However, it has been modified to work only with the `PROGRAMMED BURN` option in ALEGRA (see Section 8.5.1 on page 177). When using this `PROGRAMMED BURN JWL` equation of state model, it is also necessary to include the `PROGRAMMED BURN` input. This model is being superseded by the `KEOS JWL` model (see Section 12.3.5 on page 216), which can be used with or without the `PROGRAMMED BURN` option. Input parameters must be in consistent units, which is not usually the way `JWL` parameters are published [10]. The `JWL` constants are often cited in the following system of units: `A`, `C` and `PCJ` are given in Mbar = 10^{12} GPa, density in gm/cm^3, `DCJ` in cm/μsec, and `E0` in Mbar-cm^3/cm^3.

- Modules: material_libs/standard_models
  - progburn_jwl.h
  - progburn_jwl.C
- Physics: hydrodynamics

Table 86: Input Parameters for PROGRAMMED BURN JWL.

Parameter Name	Type	Description
RHO REF	real	Density in unreacted reference state
TREF	real	Temperature in unreacted reference state
E SHIFT	real	Arbitrary shift of reference energy (optional)
A	real	JWL parameter in units of problem
B	real	JWL parameter in units of problem
C	real	JWL parameter in units of problem
OMEGA	real	dimensionless JWL parameter
R1	real	dimensionless JWL parameter
R2	real	dimensionless JWL parameter
E0	real	JWL parameter in units of problem
PCJ	real	Chapman-Jouget pressure
DCJ	real	Chapman-Jouget detonation front velocity
TCJ	real	Chapman-Jouget temperature
PB MIN RHO	real	Defaults to 0.9*RHO REF
PB TMAX	real	Defaults to maximum of 2.*TREF or 870 K

Table 87: Registered Plot Variables of PROGRAMMED BURN JWL.

Variable Name	Type	Mode	Description
SPECIFIC HEAT VOL	real	Initialize	Initialized from Cv parameter
DENSITY	real	IOPUT	Material density
ENERGY	real	IOPUT	Specific internal energy per unit mass
PRESSURE	real	IOPUT	Pressure
TEMPERATURE	real	OUTPUT	Absolute temperature
SOUND SPEED	real	OUTPUT	Bulk sound speed
DPDRHO	real	OUTPUT	Derivative of pressure with respect to density

12.11 Material Model Examples

Numerous examples of uses of material models can be found in the Alegra Regression suite which is associated with each Alegra installation. Important examples for a user to look at in terms of material models are the 2D/comprehensive and 3D/comprehensive regression tests `soldyn_mat`, `burn_mat` and `detonation_point`. The 3D/comprehensive regression suite also has the files `soldyn_mat_si`, `burn_mat_si` for examples running in SI units. There are some material models in Alegra which are not documented in the Alegra user documentation. Such models should not be considered to be fully supported.

13 Diagnostics

To evaluate the progress of the calculation or analyze the results of a simulation, there are several diagnostic methods that are available within ALEGRA. The first method discussed in this section briefly describes an interactive menu through which calculation status information can be queried. The second method discussed in this section describes global diagnostic variables that are tallied by ALEGRA and written to the EXODUS and HISPLT databases (in addition to the registered plot variables listed with each material model). The third method describes the additional variables written to the HISPLT database, many of which are directly related to the tracer points specified in the `TRACER POINTS` section of the ALEGRA input file (see Section 5.12 on page 137).

13.1 Interactive Menu

There is a limited interactive menu in ALEGRA, in which the user can make some limited queries and stop the interactive run. The menu can be obtained by typing '`hello`' into the standard input stream. Most of these interactive menu items will be found to be non-functional. A limited set of calculational status information can be obtained by typing '!' or '`status`'. Typing '`stop`' will shutdown the calculation. For the interactive menu to be functional, the `CRT` keyword must be set to `ON` in the user input file. (The default is `CRT: ON`. See Section 4.2.2 on page 74).

13.2 Global Diagnostic Variables

Mass, momentum, and energy each obey a conservation law. ALEGRA tallies the mass, momentum and energy as a diagnostic. These tallies can be used as indicators of potential inaccuracies or errors in a simulation to the extent that the conserved quantities are not maintained. These global and material global variables are written to EXODUS, and HISPLT databases if the relevant physics is exercised by the ALEGRA calculation. The variable names are the same for all database types with the following exception. Since the HISPLT database limits the variable names to 16 characters, the root name is truncated to accommodate the component designators and the material ids while remaining with the 16-character string length limit. Material identifiers

are added to the material global variable names (prefixed with "MAT") before they are written to the EXODUS database; the material identifier is appended to the variable name as ".n" in the plot request input file for the HISPLT database. The material identifier is the integer material id specified in the MATERIAL section of the ALEGRA input file. (See Section 12.1 on page 201).

13.2.1 Time Step Tallies

The ALEGRA simulation timestep is a composite of the maximum stable timestep from numerous numerical considerations. Sometimes the timestep of an ALEGRA simulation seemingly may become unreasonably small. Usually there is a good explanation for such behavior. To assist in the diagnosis of small timesteps a set of tallies has been established that report the timesteps from each factor that can affect the overall timestep.

Table 88: Timestep Tallies for Hydrodynamics.

Global Variable Name	Explanation
TM_STEP	The actual timestep use in the ALEGRA simulation. This timestep may be smaller than the minimum of the following timesteps if the TIME STEP SCALE is non-unity.
DT_HYDRO	This is the maximum stable timestep for all of HYDRODYNAMICS. It is a composite of the following timesteps.
DT_SOUND	Courant-limit timestep based upon the material sound speed computed as if this timestep acted alone.
DT_ELASTIC	Courant-like timestep based upon the elastic wave speed computed as if this timestep acted alone. The elastic wave speed is computed from the bulk and shear moduli, B and S, as $V_{elastic} = \sqrt{\left(B + \frac{4}{3}S\right)/\rho}$
continued on next page	

continued from previous page	
DT_MATVEL	Timestep based upon the material velocity relative to the mesh computed as if this timestep acted alone. This timestep affects the simulation only if ADVECTION is enabled. The intent is to prevent material from advecting to cells other than a neighboring cell.
DT_ARTVIS	Timestep based upon ARTIFICIAL VISCOSITY limitations.
DT_VOLUME	Timestep based upon the allowed MAXIMUM VOLUME CHANGE for a mesh element in a single cycle.

13.2.2 Mass Tallies

Mass is a conserved quantity. The following table summarizes the mass tallies appearing in the ALEGRA output files.

Table 89: Mass Tallies for Region (All Physics Options).

Global Variable Name	Explanation
MASSTOT	Sum of all element masses.
MASSGAIN	Mass gain due to advection through the boundary and into the mesh. Since this tally represents a gain, it should be added to the initial mass in mass balance equations.
MASSLOSS	Mass loss due to advection through the boundary and out of the mesh and due to material discarded by the Cell Doctor. Since this tally represents a loss, it should be subtracted from the initial mass in mass balance equations.
NODEMASS	Sum of all node masses, should equal the element mass.
	continued on next page

continued from previous page	
MASSERR	Mass conservation check: present mass - (initial mass + mass gain - mass loss) This check should be negligible compared to other mass tallies. Large errors may be due to remapping errors on under-resolved meshes. The user should try increasing the mesh resolution.
MAT_MASS	The global value of the mass of material "n," where n is the integer material id.

13.2.3 Momentum Tallies

Momentum is a conserved quantity. At the present time ALEGRA does not separately tally the momentum sources or sinks due to various boundary conditions or other driving forces. The following table summarizes the momentum tallies appearing in the ALEGRA output files.

Table 90: Momentum Tallies for Dynamics and All Derived Physics Options.

Global Variable Name	Explanation
XMOM YMOM ZMOM	The x, y, and z components of the total momentum for Cartesian simulations.
RMOM ZMOM THETAMOM	The r, z, and θ components of the total momentum for cylindrically symmetric simulations.
MOM_UP_X MOM_UP_Y MOM_UP_Z MOM_DOWN_X MOM_DOWN_Y MOM_DOWN_Z	The up (positive axial direction) and down (negative axial direction) x, y, and z components of the total momentum for Cartesian simulations.
	continued on next page

continued from previous page	
MAT_MOM_X MAT_MOM_Y MAT_MOM_Z	The x, y, and z components of the momentum of material "n," where "n" is the integer material id.
MAT_MOM_U_X MAT_MOM_U_Y MAT_MOM_U_Z MAT_MOM_D_X MAT_MOM_D_Y MAT_MOM_D_Z	The up (positive axial direction) and down (negative axial direction) x, y, and z components of the momentum of material "n," where "n" is the integer material id.

13.2.4 Energy Tallies

In many problems of interest, it is often desirable to know the energy budget. How much energy is related to a given physical process? What is the value of the kinetic and internal energies? How much energy is supplied by a given source or is lost to a given sink? How fast does energy change from one form to another? ALEGRA provides the user with a detailed set of energy and power tallies to answer such questions. A global energy balance can be constructed from these tallies as follows.

$$E_{error}(t) = E_{tot}(t) - [E_{tot}(0) + E_{sources}(t) - E_{losses}(t)] = 0 \tag{13.69}$$

where

$$E_{tot}(t) = E_{int}(t) + E_{kin}(t) \tag{13.70}$$

Ideally the error in this energy balance should be zero, or at least small compared to most energy tallies. Deviations from zero are typically due to missing energy tallies, numerical inaccuracies, or perhaps too large of a time step.

Typically the HISPLT file will contain tallies at more frequent intervals compared to the EXODUS file (depending on the user specification – see the `EMIT HISPLT` and `EMIT PLOT` commands on pages 77 and 78, respectively) because it is smaller and does not contain mesh information and `PLOT VARIABLES` over the entire mesh.

The following tables summarize the various energy and power tallies. The tallies are grouped by choice of the `PHYSICS` option. Extra global power tallies marked with an asterisk (*) are omitted from the output files unless the `DETAILED ENERGY TALLIES` keyword is specified.

Table 91: Energy Tallies for Region (All Physics Options).

Global Variable Name	Explanation
ETOT PTOT*	Total of the kinetic and internal energies and rate of change.
EINT PINT*	Sum of all element internal energies and rate of change.
EINTLOSS EINTGAIN	Sum of all element internal energy changes.
EERROR* PERROR*	Energy conservation check and rate of change. This check should be small compared to other energy tallies. Large errors may be due to too large a timestep, remapping errors on under-resolved meshes, or incompletely tallied sources, sinks, or boundary conditions. The user should try decreasing the timestep or increasing the mesh resolution.
MAT_EINT	The global internal energy of a material, where the material id designated by appending the material id to the variable name.
MAT_ETOT	The global total energy of a material, where the material id designated by appending the material id to the variable name.

Table 92: Energy Tallies for Dynamics (Hydrodynamics).

Global Variable Name	Explanation
EKIN PKIN*	Kinetic energy and rate of change.
EPDV* PPDV*	Time-integrated PdV energy and instantaneous rate of change. The PdV energy is the time-integrated work done by the pressure on the material. This work will manifest itself as internal energy. This energy is recoverable. If the rate of change is positive the internal energy is increasing, otherwise if the rate of change is negative, the internal energy is decreasing.

continued on next page

continued from previous page	
ENONPDV* PNONPDV*	Time-integrated non-PdV energy and instantaneous rate of change. The non-PdV energy is the time-integrated work done by the artificial viscosity and hourglass forces on the material. This work will manifest itself as internal energy. The rate of change should be positive since these are dissipative forces and the internal energy should be increasing.
EVELBC PVELBC*	Work done on the system by various kinematic boundary conditions and rate of change.
EGRAV PGRAV*	Gravitational potential energy and rate of change. The tally is included only if a non-zero GRAVITY option is specified. Zero potential energy is defined to be at the origin.
EINTGAIN EINTLOSS	Internal energy gain or loss due to advection through the boundary of the mesh and internal energy discarded by the CELL DOCTOR. Energy gain is plotted as a positive value and energy loss is plotted as a negative value.
EKINGAIN EKINLOSS	Kinetic energy gain or loss due to advection through the boundary of the mesh and kinetic energy discarded by the CELL DOCTOR. Energy gain is plotted as a positive value and energy loss is plotted as a negative value.
EK_UP_X EK_UP_Y EK_UP_Z EK_DOWN_X EK_DOWN_Y EK_DOWN_Z	Partial kinetic energies associated with each coordinate axis and direction (D = down, U = up).
MAT_EK	The global kinetic energy of a material, where the material id designated by appending the material id to the variable name. In EXODUS, the id is appended to the end of the variable name in the plotting database. For HISPLT, the material id is appended by the user in the plot request within the HISPLT input file.
continued on next page	

continued from previous page	
MAT_EK_UP_X MAT_EK_UP_Y MAT_EK_UP_Z MAT_EK_DOWN_X MAT_EK_DOWN_Y MAT_EK_DOWN_Z	Partial material kinetic energies associated with each coordinate axis and direction (D = down, U = up). The material id is designated by appending the material id to the variable name. In EXODUS, the id is appended to the end of the variable name in the plotting database. For HISPLT, the material id is appended by the user in the plot request within the HISPLT input file.

13.2.5 Additional Diagnostic Variables

Additional variables are written to the EXODUS and HISPLT databases. The grind time is useful for comparing the relative effort of a calculation per element, per cycle. This is calculated with and without the time required for reading and writing all input and output files. The CPU time is also provided, excluding the I/O time.

Table 93: Global Variables In Addition to Energy/Mass/Momentum Tallies.

Global Variable Name	Explanation
GRIND	Grind time during the calculation, defined as the computation time divided by the product of the number of processors X number of elements X number of cycles.
CPUNOIO	CPU time ignoring the contribution of IO.
GRINDNOIO	Grind time ignoring the contribution of IO.

13.3 Additional HISPLT Database Variables

In addition to all global and material global variables listed in Section 4.2.12 on page 81, the HISPLT database contains variables relevant to the tracer points. History variables calculated at tracer particle locations include all the plot variables specific to the material models assigned to each material (see the plot variable tables in the material MODEL input section) and general output variables listed below (Table 94).

Hisplt input files can be constructed by referring to these variable lists or by using the CATALOG input keyword in an initial run of HISPLT to obtain the list of variables specific to the database being read.

Table 94: Point History Variables.

Global Variable Name	Explanation
REGION-ID	Only one region exists for ALEGRA at this time.
BLOCK-ID	The block id for the tracer particle.
ELEMENT-ID	The global element id within which the tracer particle resides.
ON-OFF	In a parallel calculation, number of processors that contain the coordinates of this tracer location. (This variable is currently inactivated but will be enabled in a future release.)
PSY-X PSY-Y PSY-Z	Position of the tracer within the local coordinate system of the element on which the tracer resides.
ELEMENT-VOLUME	Volume of the element within which the tracer particle resides.
ELEMENT-MASS	Mass of the element within which the tracer particle resides.
XPOSITION	X position of the tracer particle resides.
YPOSITION	Y position of the tracer particle resides.
ZPOSITION	Z position of the tracer particle resides.

For the structured mesh option, the ***ELEMENT-ID*** *that is returned is not easily interpreted since it is an actual index into the block element array.*

Table 95: Global History Variables Specific to HISPLT.

Global Variable Name	Explanation
TIME	The solution time in the calculation.
DT	Time step size in the calculation.
CPU	Accumulated computation time during the calculation.
CYCLE	Accumulated number of cycles during the calculation.

14 Performance Measurement in ALEGRA

The large memories and processor capability of massively parallel processor (MPP) computers allows the analysis of large two and three-dimensional problems. These problems may take hours to days even on very large parallel computers and it is important to achieve the solution times as short as possible. This high-performance requirement is not easy to satisfy as there are various levels of granularity and memory hierarchies which should be considered to achieve high performance and the design of the code and algorithms can significantly affect performance. One of the first steps to be taken in the effort to take performance seriously is to provide standardized means for measuring performance, measure this performance on a regular basis, and then measure the improvement over gradual refactoring of algorithms and coding.

The only truly relevant measure of performance in scientific computing is the time-to-solution to achieve a specified level of accuracy on a given set of resources. Implicitly this means that both algorithms and machine efficiency must be considered to obtain a balanced view on performance. Historically, algorithms have been just as crucial as machine speed in improving performance. For expediency sake, more simple measures are often used. Useful metrics include wall clock time, memory usage, compute/communicate ratios, FLOPS (floating point operations/second), "grind time" (second/cycle/cell) and scalability curves. Generally these measure are useful but not sufficient to gain a balanced picture. For example, a simple iterative solve may achieve a much higher FLOPS rating than a multigrid method, but a multigrid method may get to the correct solution much faster. Very poor FLOPS numbers may be an indication that the coding style or organization is not cache friendly and that too much time is being spent in memory movement to and from main memory or in off processor communications. Inefficient computations might also lead to excellent scalability numbers while masking inefficiencies in the communications. Generally a user will want to look at parallel scalability by increasing the number of processors while at the same time increasing the size of their problem. Optimal scaling implies that the wall clock time will be roughly constant. This linear scalability is easy to achieve for algorithms which are only local in nature but is much more difficult to achieve for problems which include global interactions or multiple physics with competing demands. The ability to achieve scalability depends also on the number of processors and the speed of the communication fabric relative to the processor speed.

The primary mechanism for performance measurement in ALEGRA is to turn on the various debug mode profiling options. First, outfile requests must be turned on. For example, `EMIT OUTPUT, CYCLE INT 1`. Then, profiling output must be turned on, such as `DEBUG MODE=PROFILETIME`, `DEBUG MODE=PROFILEMEMORY` or `DEBUG MODE=PROFILEHARDWARE`. These turn on specific timers for various sections of the code and the results finally end up in the .out file. This may be helpful while working with a developer to pinpoint specific issues and performance bottlenecks. `PROFILETIME` and `PROFILEMEMORY` are supported on all platforms and give timing and memory usage. PROFILEMEMORY may cause some slowdown due to issues of frequent access to the allocator information routines. `PROFILEHARDWARE` is supported only on some platforms and gives information such as floating point operation rates. The collected data may be machine specific, and on platforms that do not support the collection of hardware performance counters, there is no output. The output comes out twice, once sorted in a hierarchical manner according to nested calls, and once with a flat alphabetically sorted output list by tag name. The MPE message passing profiling library is available for some platforms and should be discussed with a developer if such access is desired.

14.1 Tricks and Traps

This section is intended to help the user start thinking about how to get the maximum performance from Alegra on their own machine set.

- Define only the materials you need in the input deck. There is a small but real cost associated with materials that are defined, even if they are never used in the problem. Some users tend to define all materials they may ever use in the input deck, and use only a subset of them in any given iteration of the problem. Commenting out these material definitions will result in a slightly smaller memory footprint of the Alegra mesh, and slightly faster execution as well.

- Try to size and shape your mesh so that when it is spread, the mixed material regions will cross as few processor boundaries as possible during the run.

- The surface area to volume ratio of each processor subdomain has a significant impact on performance. Large fractions of elements involved

in ghost mesh updates will very significantly impact MPP performance. The default mesh partitioner will try to minimize the surface area to volume ratio of elements on the surface relative to elements in the interior.

- In general for most MPP machines, it will greatly benefit machine efficiency in Alegra to maximize the amount of work on a node. Typically this means that you will size and spread the mesh so that every node has equal number of elements, and that roughly 90% of physical memory is used for the ALEGRA calculation.

- Some machines have more than one processor per node and these processors may be required to be involved with message passing implementations. It may be preferable to run N - 1 processes per node rather than use all processors per node essentially leaving one processor free for system tasks and message passing interrupts. (For instance if your machine has 4 processors per node, use 3 rather than 4). On some machines this may not matter very much and using all processes on all nodes with memory usage at $\tilde{9}0\%$ may give the best performance.

- For structured mesh physics use an inline mesh specification or translate a genesis file into a plot3d file outside of ALEGRA. Using the genesis input format places a significant memory burden on processor 0.

- Limit the amount of output requested via both frequency selection and the number of variables.

- Run Lagrangian if at all possible.

- Try running solid dynamics and hydrodynamics problems using the structured mesh physics options. The structured mesh implementation runs much faster than the unstructured mesh implementation.

15 Frequently Asked Questions

If I am familiar with running CTH*, how do I run a similar problem with* ALEGRA*?*

The first major difference between CTH and ALEGRA is the mesh input. In addition to the unstructured mesh physics that has been ALEGRA's traditional mesh, ALEGRA now has a choice of using structured mesh physics, which is more efficient but supports only a subset of the unstructured code's capabilities.

For unstructured mesh physics, the mesh must be generated outside of ALEGRA. This allows ALEGRA to use sophisticated unstructured meshes for Lagrangian and ALE calculations, as well as simple unstructured meshes similar to those used in CTH. The CTH mesh must be an orthogonal structured mesh, so it is a simple thing to generate the mesh in CTHGEN [1]. In ALEGRA, the mesh is generated separately by another program. Examples of such programs include FASTQ (2D), a combination of FASTQ and GEN3D (3D), or the CUBIT mesh generation tool (2D and 3D).

You can insert materials into the mesh just like in CTH using the `DIATOM` capability (see the above ALEGRA documentation for slight differences, most importantly in the keywords). Materials can also be assigned on a block-by-block basis. Each "`REGION`" created by FASTQ corresponds to a "`BLOCK`" in ALEGRA. Individual blocks are assigned material numbers in the ALEGRA block input. (Note that one can still use `DIATOM` to insert material into an empty unstructured element block).

FASTQ and GEN3D are part of ACCESS. (Make sure your ALEGRA environment is specified in your path before the ACCESS paths, since at least one of the files has a duplicate name). Complete FASTQ and GEN3D documentation should be provided with your ACCESS distribution.

For structured mesh physics, the mesh input can either be specified in the input file (see Section 11), or the mesh can be generated externally as long as each block has the typical CTH ijk order (e.g., in each block, the number of cells in the y and z directions are constant for all x). If the mesh is generated externally by a program other than PLOT3D, ALEGRA will translate the mesh to the PLOT3D format for use in the structured mesh physics code.

Mesh generation for ALEGRA consists of only the geometric discretization

and the assignment of sideset and nodeset flags that may be referenced by boundary conditions or certain physics packages in the ALEGRA run. Whereas CTHGEN assigns materials and material properties prior to the CTH run, these functions are performed in a single ALEGRA run. Sample input files for both mesh generation and ALEGRA can be found in the Benchmark directories.

What does "Segmentation violation" mean?

This is the message used by the UNIX operating system to tell you that your program has tried to access memory that has not been assigned to it. It is always the result of a bug if you see it while running ALEGRA. Such bugs should be reported to the development team promptly with an associated sample input file and mesh.

How seriously should I take warnings?

A warning indicates a condition when initializing or running the code that one of the ALEGRA code developers thinks is suspicious, but not necessarily wrong or perhaps not worth stopping the calculation since it may still be possible to get useful results. Since ALEGRA is very good at solving a discrete representation of your physics equations, but not very good at evaluating the results, it only notices the most obvious problems, such as negative temperatures. Warnings should therefore be taken seriously and investigated. However, after issuing a warning, ALEGRA will continue the calculation on the assumption that the problem is transitory and will not spoil the calculation as a whole.

My simulation seems to be taking forever to start up. What is going on?

By default, ALEGRA will copy the entire contents of the input file to the information data records of the EXODUS [29] output file. For very large input files, *e.g.*, those having long `FUNCTION` definitions or large numbers of `DIATOM PACKAGE`s, this can be slow and tedious and result in long initialization times. One suggestion is to reduce the amount of input data if possible, *e.g.*, limit the number of data points in a `FUNCTION` table. If this is not possible, then the default behavior can be suppressed and initialization times can be shortened by specifying `COPY INPUT = FALSE` in your input file. See Section 4.2.1 on page 74.

My calculation reports an "element inversion." What does this mean?

ALEGRA has detected an element with a negative volume, indicating that

the element has turned inside out. If this occurs right away, it may simply mean that one of the initial conditions you specified is causing an element to collapse before it has a chance to respond. This can often be corrected by specifying a smaller `MAXIMUM INITIAL TIME STEP` (see Section 5.6.2 on page 98).

Element inversions later in a calculation usually arise from one of two causes. If the element inversion is preceded by a sharp drop in the time step, it is usually the result of mesh tangling. Susceptibility to such tangling is a weakness of the quadrilateral or hexahedral elements used in most ALEGRA calculations. The usual way to fix this is to identify the portion of the mesh that is tangling and make it part of an Eulerian or ALE element block. (Using a triangular or tetrahedral mesh is usually NOT a good way to fix the problem, since these elements are numerically very stiff.)

Abrupt element inversion (with no warning signs preceding it) usually indicates either that a large amount of energy has suddenly been deposited in an element, or a material in the element has run off the bounds of an equation of state table. The `CLIP` option for the Kerley `SESAME` equation of state can be useful (see Section 12.3.7 on page 223.) If no explanation for this behavior can be found, it may indicate a program bug that should be reported to the development team.

The time step has become so small that the calculation is going nowhere. What do I do?

Most of the problems that lead to element inversion can also lead to a sharp drop in the time step. In addition, a very hot material in an element can give rise to a large sound speed and a very small stable time step. The `CELL DOCTOR` package (see Section 5.11 on page 135) can detect and remove tiny volumes of very hot material that sometimes are left in an otherwise cold or empty cell by the advection package.

ALEGRA is also known to use a very conservative time step calculation. There are several user-controllable features that can affect the time step calculation. One is the `MINIMUM ELEMENT SIDE TIMESTEP CONTROL` (see Section 8.4.4 on page 176) which controls the calculation of the characteristic length for the element used in the time step calculation. Another parameter is the `MAXIMUM VOLUME CHANGE` (see Section 8.4.3 on page 176) value which can often cause ALEGRA to use very small time steps in Eulerian calculations. Finally, the `TIME STEP SCALE` (see Section 5.6.6 on page 99) parameter con-

trols the overall multiplier that ALEGRA applies to the computed time step to determine what value to actually use in the calculation.

*I am using **ALE** in my calculation and I get an almost immediate abort with a message in the output that says:*

```
> Error: Dynamics::Determine_Volume_Fluxes()
>   Element = 864 Volume Flux = 1.39057e-11 is greater than the
>   volume = 5.1859e-12
```

What's happening here is that ALEGRA's idea of a "good" mesh and the mesh generator's idea of good do not agree, so ALEGRA rezone is doing its own thing. In the course of doing that, it is moving the nodes too far in one step and resulting in an "overflux" where the amount of "space" moved out of an element exceeds the amount there to start. What needs to happen is to let ALEGRA do its thing, but do it more gradually. There are inputs in Domain called

```
initial remesh movement limiter [real]
remesh movement ratio [real]
remesh movement limiter [real]
```

The first two commands set an initial limit multiplicative factor on the movement of any node and then a growth factor that is applied every remesh pass (usually remesh is done 10 times every cycle). The last value is an ultimate limit to which the remesh limiter will grow (has to be <= 1.0). So what you could do is put in the domain section:

```
initial remesh movement limiter = 0.10
remesh movement ratio           = 1.05
remesh movement limiter         = 1.0
```

so the limit factor will start at 10%, grow by 5% per application and stop at 1.0.

My run is stopping before it ever gets to the first cycle print. I have no idea what is going wrong. What information should I gather to make it easier to find out what is going on?

Use the DEBUG MODE print capability to help locate where the code is dying. As described in Section 4.2.3 on page 75, the "LOCATION" debug flag will produce a trail on standard out that will help us track down where the code is dying. If you are going to send us a problem to take a look at, we usually need the *.inp and any file used to generate the *.gen file: FASTQ, GEN3D input files or CUBIT journal files. Any special input files, like volume fraction files for DIATOM, are also needed.

My run aborts before anything seems to happen. The screen message says to look at the .out file, but all I see in that file is the following.

```
Error: Volume_Average_Plot_Expression::Set_Up
       Cannot find a material variable named VELOCITY. See the *.out
       file for valid variable names.
Error: Region::Set_Up
       Unable to set up plot database
```

You may also see this occur for variables other than VELOCITY. In your input file, you have a plot variable specified with the "avg" modifier after the name. This tells ALEGRA to use a volume-weighted average of the value of this quantity over all materials in an element and report one value for the element, as opposed to reporting one value for each material in the element. The problem is that the variable, VELOCITY in this particular case, is not a "material" variable. Remove the "avg" modifier and your run will proceed.

How do I handle common nodes between two or more non-orthogonal no displacement boundary conditions?

Caution should be exercised whenever two or more NO DISPLACEMENT sidesets share a common edge or vertex. The NO DISPLACEMENT boundary condition (Section 7.1.2 on page 145) operates by removing the normal component of the nodal acceleration vector for all the nodes on the sideset. This application occurs sideset by sideset in the order that the sidesets are specified in the input deck. When two non-orthogonal sidesets share a common edge, the application of the second sideset will result in the nodes along the common edge having an acceleration component that is normal to the first sideset. It is important to recognize that this problem does not arise if the two sidesets are orthogonal to each other. This is actually the key to working around this problem.

When two non-orthogonal sidesets meet, a nodeset should be defined

to contain the nodes that lie on the common edge between the `sidesets`. Realize that the `NO DISPLACEMENT sidesets` already assumes that the two `sidesets` are planar so their common edge must be a line and the nodes along the common edge must be constrained only to move along the edge. This is accomplished by specifying `NO DISPLACEMENT` boundary conditions over both the `sidesets` as is usually done in the input deck. To constrain the edge nodes, however, a third `NO DISPLACEMENT` boundary condition must be specified for the edge node set so that they do not move in the direction

$$(\hat{n}_1 \times \hat{n}_2) \times \hat{n}_2 \tag{15.71}$$

where $\hat{n}_1$ and $\hat{n}_2$ are the normal directions specified for the first and second `sidesets` as they appear in the input deck respectively. Note that the order of the `sidesets` is important. After ALEGRA applies the second `sidesets`, the acceleration vector for the nodes along the edge will lie in the plane of the second `sidesets`. The cross product within the parenthesis is a vector tangent to the edge. By removing the acceleration component in the direction of the cross product of the tangent vector and the normal to the second surface, the resulting acceleration vector must act along the edge. The following ALEGRA input records provide an example where `nodeset 1` is the common edge between `sidesets 10` and `20`:

```
no displacement, sideset 10, normal, x  0.50 y 0.0  z  1.0
no displacement, sideset 20, normal, x  0.50 y 1.0  z  0.0
no displacement, nodeset  1, normal, x -0.50 y 0.25 z -1.25
```

Another special case is a vertex node that is shared by three non-coplanar `NO DISPLACEMENT sidesets`. In this case, the vertex node must be placed in another `nodeset` and constrained to have a zero acceleration since it cannot move in any direction.

References

[1] R. L. Bell et al. CTHGEN user's manual and input instructions. Technical report Version 5.0, Sandia National Laboratories, Albuquerque, NM, 2002. unpublished.

[2] D. J. Benson. A new two-dimensional flux-limited shock viscosity for impact calculations. *Computer Methods in Applied Mechanics and Engineering*, 93:39–95, 1991.

[3] T. D. Blacker. FASTQ Users Manual Version 1.2. Technical report SAND88-1326, Sandia National Laboratories, Albuquerque, NM, July 1988.

[4] R. M. Brannon and M. K. Wong. MIG Version 0.0 Model Interface Guidelines: Rules to Accelerate Installation of Numerical Models Into Any Compliant Parent Code. Technical report SAND96-2000, Sandia National Laboratories, Albuquerque, NM, August 1996.

[5] K. H. Brown et al. ACME Algorithms for Contact in a Multiphysics Environment API Version 1.3. Technical report SAND2001-1470, Sandia National Laboratories, Albuquerque, NM, May 2003.

[6] K. H. Brown et al. CavityExpansion: A library for cavity expansion algorithms, version 1.0. Technical report SAND2003-1048, Sandia National Laboratories, Albuquerque, NM, 2003.

[7] R. Brun and L. D. Dumitrescu, editors. *CTH: A Software Family for Multi-Dimensional Shock Physics Analysis*, Marseille, France, July 1993. Proceedings of the 19th International Symposium on Shock Waves Vol. I.

[8] K. G. Budge and J. S. Peery. RHALE: A MMALE shock physics code written in C++. *International Journal of Impact Engineering*, 14:107–120, 1993.

[9] J. Chakrabarty. *Theory of Plasticity.* McGraw-Hill Book Company, New York, 1987.

[10] B. M. Dobratz and P. C. Crawford. LLNL explosives handbook. Technical report UCRL-52997, Lawrence Livermore National Laboratory, Livermore, California, January 1985.

[11] M. A. Ellis and B. Stroustrup. *The Annotated C++ Reference Manual*. Addison-Wesley Publishing Company, Reading, MA, 1990.

[12] D. P. Flanagan and T. Belytschko. A uniform strain hexahedron and quadrilateral with orthogonal hourglass control. *International Journal for Numerical Methods in Engineering*, 17:679–706, 1981.

[13] A. F. Fossum and R. M. Brannon. The SANDIA GEOMODEL Theory and User's Guide. Technical report SAND2004-3226, Sandia National Laboratories, Albuquerque, NM, 2004.

[14] A. F. Fossum and J. T. Fredrich, editors. *Cap Plasticity Models and Compactive and Dilatant Pre-Failure Deformation*, 2000. Pacific Rocks 2000.

[15] A. P. Gilkey and J. H. Glick. BLOT - A Mesh and Curve Plot Program for the Output of a Finite Element Analysis. Technical report SAND88-1432, Sandia National Laboratories, Albuquerque, NM, June 1989.

[16] A. P. Gilkey and G. D. Sjaardema. GEN3D: A GENESIS Database 2D to 3D Transformation Program. Technical report SAND89-0485, Sandia National Laboratories, Albuquerque, NM, March 1989. Fourth Printing February 1994.

[17] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.

[18] William D. Gropp and Ewing Lusk. *User's Guide for* `mpich`*, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[19] B. Hendrickson and R. Leland. The Chaco users guide version 1.0. Technical report SAND93-2339, Sandia National Laboratories, Albuquerque, NM, 1993.

[20] G. L. Hennigan, M. St. John, and J. N. Shadid. NEMESIS I: A set of functions for describing unstructured finite-element data on parallel computers. Technical report, Sandia National Laboratories, Albuquerque, NM, May 1998.

[21] G. I. Kerley. CTH reference manual: The equation of state package. Technical report SAND91-0344, Sandia National Laboratories, Albuquerque, NM, 1991.

[22] G. I. Kerley. CTH reference manual: The equation of state package. Technical report SAND98-0947, Sandia National Laboratories, Albuquerque, NM, 1998.

[23] G. I. Kerley. Recent improvements to the CTH EOS package. Technical report KPS99-1, Kerley Publishing Services, March 1999.

[24] J. M. McGlaun. CTH reference manual: Cell thermodynamics. Technical report SAND91-0002, Sandia National Laboratories, Albuquerque, NM, 1991.

[25] J. M. McGlaun, S. L. Thompson, and M. G. Elrick. A brief description of the three-dimensional shock wave physics code CTH. Technical report SAND89-0607, Sandia National Laboratories, Albuquerque, NM, 1989.

[26] J. S. Peery and D. E. Carroll. ALE remap algorithms in ALEGRA. Unpublished technical report, Sandia National Laboratories, Albuquerque, NM, 1999.

[27] M. H. Rice, R. G. McQueen, and J. M. Walsh. *Advances in Research and Applications: Solid State Physics*, volume 6, chapter Compression of Solids by Strong Shock Waves. Academic Press, 1958.

[28] A. C. Robinson, J. R. Weatherby, and J. B. Aidun. Periodic boundary conditions in the ALEGRA finite element code. Technical report SAND99-2698, Sandia National Laboratories, Albuquerque, NM, November 1999.

[29] L. A. Schoof and V. R. Yarberry. EXODUS II: A Finite Element Data Model. Technical report SAND92-2137, Sandia National Laboratories, Albuquerque, NM, November 1995.

[30] S. A. Silling. CTH reference manual: Viscoplastic models. Technical report SAND91-0292, Sandia National Laboratories, Albuquerque, NM, 1991.

[31] S. A. Silling. Brittle failure kinetics model for concrete: Structures under extreme loading conditions. *ASME: PVP*, 351:263–268, 1997.

[32] G. D. Sjaardema. APREPRO: An Algebraic Preprocessor for Parameterizing Finite Element Analyses. Technical report SAND92-2291, Sandia National Laboratories, Albuquerque, NM, December 1992. Updated March 1997.

[33] G. D. Sjaardema. GJOIN: A Program for Merging Two or More GENESIS Databases. Technical report SAND92-2290, Sandia National Laboratories, Albuquerque, NM, December 1992.

[34] G. D. Sjaardema. GROPE: A GENESIS/EXODUS Database Examination Program. Technical report SAND92-2289, Sandia National Laboratories, Albuquerque, NM, December 1992.

[35] G. D. Sjaardema. Overview of the Sandia National Laboratories Engineering Analysis Code Access System. Technical report SAND92-2292, Sandia National Laboratories, Albuquerque, NM, January 1993. Reprinted August 1994.

[36] G. D. Sjaardema et al. CUBIT mesh generation environment, vol. 2: Developers manual. Technical report SAND94-1101, Sandia National Laboratories, Albuquerque, NM, 1994.

[37] R. M. Summers et al. Recent progress in ALEGRA development and application to ballistic impacts. *International Journal of Impact Engineering*, 20:779–788, 1997.

[38] L. M. Taylor and D. P. Flanagan. PRONTO-2D: A two-dimensional transient solid dynamics program. Technical report SAND86-0594, Sandia National Laboratories, Albuquerque, NM, March 1987.

[39] L. M. Taylor and D. P. Flanagan. PRONTO3D: A three-dimensional transient solid dynamics program. Technical report SAND87-1912, Sandia National Laboratories, Albuquerque, NM, March 1989.

[40] P. A. Taylor. CTH reference manual: The steinberg-guinan-lund viscoplastic model. Technical report SAND92-0716, Sandia National Laboratories, Albuquerque, NM, 1992.

[41] P. A. Taylor. CTH reference manual: The bammann-chiesa-johnson viscoplastic/damage model. Technical report SAND96-1626, Sandia National Laboratories, Albuquerque, NM, 1996.

[42] S. L. Thompson and L. N. Kmetyk. HISPLT, A Time-History Graphics Postprocessor Users' Guide. Technical report SAND91-1767, Sandia National Laboratories, Albuquerque, NM, September 1991. Revised April 1994.

[43] J. M. Walsh and R. H. Christian. Equation of state of metals from shock wave measurements. *Physical Review*, 97(6):1544–1556, 1955.

Index

DISTRIBUTION:

2 Oak Ridge National Laboratory
P.O. Box 2009,
Oak Ridge, TN 37831
J. Michael Starbuck, MS 8048
Seokho Kim, MS 8045

1 Vanden, K. J.
AFRL/MNAC
101 West Eglin Blvd.
Eglin AFB, FL 32542-6810

1 Chin, Chuck
U.S. Army ARDEC
SMCAR-AEE-WW -
Bldg. 3022
Picatinny Arsenal, NJ 07806-5000

1 Doney, Bobby
U.S. Army Research Laboratory
AMSRL-WT-TA
Aberdeen Proving Ground, MD, 21005-5066

1 Fermen-Coker, M.
U.S. Army Research Laboratory
AMSRL-WT-TC
Aberdeen Proving Ground, MD 21005-5066

1 Filbey, Gordon L., Jr.
U.S. Army Research Laboratory
AMSRL-WT-TC
Aberdeen Proving Ground, MD 21005-5066

1 Kimsey, Kent D.
U.S. Army Research Laboratory
AMSRL-WT-TC
Aberdeen Proving Ground, MD 21005-5066

1 Kingman, Pat
U.S. Army Research Laboratory
AMSRL-WT-TD
Aberdeen Proving Ground, MD 21005-5066

1 Schraml, Stephen J.
U.S. Army Research Laboratory
AMSRL-WT-TC
Aberdeen Proving Ground, MD 21005-5066

1 Becker, J. D.
Attn: CEERD-IH-N
U.S. Army Engineering & Research Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

1 Howard, Scott
US Army AMCOM
AMSAM-RD-PS-WF
Bldg 5400
Redstone Arsenal, AL 35808

1 Poy, Stephen
Code 673
Carderock Division
Naval Surface Warfare Center
9500 MacArthur Blvd.
West Bethesda, MD 20817-5700

1 Sorenson, Carl D.
Brigham Young University
Dept. of Mechanical Engineering
435 CTB
Provo, UT 84602

1 Ding, Jow L.
Institute for Shock Physics
Washington State University
P.O. Box 642814
Pullman, WA 99164-2814

1 Haque, Aamer
U. S. Deptartment of Energy
MS DP-153
1000 Independence Ave. S. W.
Washington, DC 20585

1 Moore, Timothy W.
600 Boulevard South
Suite 106
Huntsville, AL 35806

1 Stecher, Frederick
Alliant TechSystems
MS: MN07-MW44
4700 Nathan Lane, North
Plymouth, MN 55442-2512

1 Stults, Allen
ITT Industires Advanced Engineering & Sciences
600 Boulevard South, Suite 208
Huntsville, AL 35802-2104

1 Scott, Daniel
SAIC
1710 SAIC Dr.
M/S T1-5-2
McLean, VA 22102

1 Bassler, Robert
Applied Research Assoc.
Capital Area Division
2760 Eisenhower Ave., Suite 308
Alexandria, VA 22314-4569

1 MS 0139
P. Yarrington, 9902

1 MS 0321
W. J. Camp, 9200

1 MS 0370
P. N. Demmie, 9232

1 MS 0370
A. V. Farnsworth, Jr., 9232

1 MS 0370
M. E. Kipp, 9232

1 MS 0370
T. G. Trucano, 9211

1 MS 0378
W. J. Bohnhoff, 9231

1 MS 0378
S. Carroll, 9231

1 MS 0378
M. A. Christon, 9231

1 MS 0378
R. R. Drake, 9231

1 MS 0378
D. M. Hensinger, 9231

1 MS 0378
D. A. Labreche, 9231

1 MS 0378
C. B. Luchini, 9231

1 MS 0378
S. J. Mosso, 9231

1 MS 0378
S. V. Petney, 9231

1 MS 0378
A. C. Robinson, 9231

1 MS 0378
J. Robbins, 9231

1 MS 0378
R. M. Summers, 9231

1 MS 0378
T. E. Voth, 9231

1 MS 0378
M. K. Wong, 9231

1 MS 0378
R. L. Bell, 9232

1 MS 0378
G. C. Bessette, 9232

1 MS 0378
R. A. Cole, 9232

1 MS 0378
S. A. Silling, 9232

1 MS 0378
P. A. Taylor, 9232

1 MS 0521
S. T. Montgomery, 2561

1 MS 0557
T. W. Simmermacher, 9124

1 MS 0826
J. D. Zepper, 9143

1 MS 0835
J. M. McGlaun, 9140

1 MS 0835
E. A. Boucheron, 9141

1 MS 0835
M. W. Glass, 9141

1 MS 0836
D. A. Crawford, 9116

1 MS 0836
E. S. Hertel, Jr., 9116

1 MS 0836
R. G. Schmitt, 9116

1 MS 1110
P. B. Bochev, 9214

1 MS 1110
J. B. Aidun, 9230

1 MS 1152
S. J. Chantrenne, 1642

1 MS 1152
R. S. Coats, 1642

1 MS 1152
M. L. Kiefer, 1642

1 MS 1152
J. D. Kotulski, 1642

1 MS 1152
L. P. Mix, Jr., 1642

1 MS 1152
M. F. Pasik, 1642

1 MS 1152
T. D. Pointon, 1642

1 MS 1152
D. B. Seidel, 1642

1 MS 1152
C. D. Turner, 1642

1 MS 1166
C. R. Drumm, 15345

1 MS 1166
W. C. Fan, 15345

1 MS 1152
S. J. Chantrenne, 1642

1 MS 1168
C. Deeney, 1646

1 MS 1181
J. P. Davis, 1646

1 MS 1186
T. A. Brunner, 1674

1 MS 1186
R. B. Campbell, 1674

1 MS 1186
P. J. Christenson, 1674

1 MS 1186
K. R. Cochrane, 1674

1 MS 1186
M. P. Desjarlais, 1674

1 MS 1186
C. J. Garasi, 1674

1 MS 1186
T. A. Haill, 1674

1 MS 1186
R. J. Lawrence, 1674

1 MS 1186
R. W. Lemke, 1674

1 MS 1186
T. A. Mehlhorn, 1674

1 MS 1186
T. K. Mattsson, 1674

1 MS 1186
B. V. Oliver, 1674

1 MS 1186
K. Peterson, 1674

1 MS 1186
S. A. Slutz, 1674

1 MS 1186
R. A. Vesey, 1674

1 MS 1186
E. C. Wemlinger, 1674

1 MS 1186
E. P. Yu, 1674

1 MS 1190
J. P. Quintenz, 1600

1 MS 1191
M. K. Matzen, 1670

1 MS 1191
M. A. Sweeney, 1670

1 MS 1194
S. E. Rosenthal, 1644

1 MS 1194
E. M. Waisman, 1644

1 MS 1452
S. L. Szarka, 2552

1 MS 9051
S. E. Wunsch, 8351

1 MS 9159
J. J. Hu, 9214

1 MS 9159
R. S. Tuminaro, 9214

2 MS 0899
Technical Library, 9616

1 MS 9018
Central Technical Files, 8945-1